プログラマーのための

X68000

環境(エンバイロメント)ハンドブック

吉沢正敏
市原昌文

工学社

コンピュータの中に世界があります．
未到の砂漠，広大な草原，世界を支える宇宙樹，etc….
MPUの中に，メモリの中に，ディスクの中に，
さまざまな風景を見ることができます．
X68000という豊穣な大地を取り巻くHuman 68k/IOCSという環境の中，
あなたはどんな種を蒔きますか？
どんな生命を育みますか？

プログラマーのためのX68000環境

目次

❖第1部Human68K❖❖

ハンドブック

吉沢正敏・市原昌文／著

❖第2部IOCSコール❖❖

❖付 録❖

本書の使い方

本書はX68000シリーズの標準OS，Human 68k環境でアプリケーションを開発するために必要な情報を，

①目的の情報をすばやく
②関連事項を有機的に
③適切な使用例を豊富に

というコンセプトのもとにまとめたものです．

第1部ではHuman 68k Ver 1.0x，Human 68k Ver 2.0xの内部操作およびファンクションコールについて吉沢が解説しています．第2部ではIOCSについて市原が解説しています．

1部，2部ともアセンブラで開発を行なうことを前提として書かれていますが，C言語でライブラリを作成する場合などにも役立てて頂けると思います．

第1部

Human68K

1章 Humanの内部動作

Human68k（以下 Human）のアプリケーションを作製するためにはファンクションコールとあわせて Human 内部の動作も理解しておく必要があります．

この章では，特徴的な Human の内部動作を解説して，その後で個々のファンクションについて解説していくことにします．

1.1 メモリ／プロセス管理

Human においては，その上で走るプログラムのことを「プロセス」と呼びます．プロセスの中から別のプロセスを呼び出して実行させることもできますが，この場合，呼び出した側のプロセスを「親プロセス」，呼ばれた側を「子プロセス」と呼びます．子プロセスからさらにプロセスを起動した場合は，「孫プロセス」ということになります．

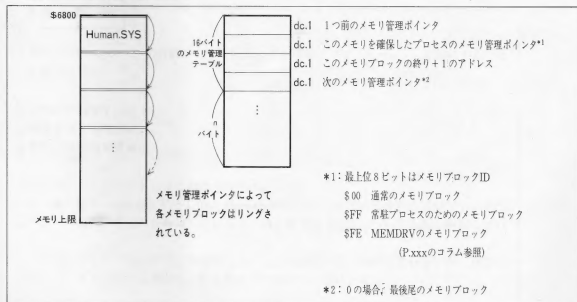
このように複数のプロセスをメモリに読み込んで実行するために，OS は現在のメモリの使用状態を把握し，管理する必要が出てきます．

Human では「メモリ管理テーブル」と「プロセス管理テーブル」（PSP）によってメモリとプロセスを管理しています．

□ 1.メモリ管理の概要

プロセスをいくつもメモリ上に置くためには，各プロセスの存在するエリアや，ワークエリアが重なってしまうことは許されません．

そのために，プロセスは，ワークエリアなどに使うための領域を$FF48 mallocファンクションなどを使ってHumanに請求し，不用になったら$FF49 mfreeなどで開放してHumanに返してやることになります．*註 Humanに管理される限りはプロセス間でメモリを競合して使うという事態は発生しません．

Fig.1.1.1　Human のメモリ管理

プロセスからメモリの請求がくると，Humanはそれだけのメモリを確保できる領域がある場合は，その領域に「メモリ・ブロック」を形成し，プロセスに管理を任せます．このとき，プロセスには「メモリ管理テーブル」へのポインタが返されます．

メモリ・ブロックはFig.1.1.1のような構成になっていて，その先頭にはつねに4ロングワード(16バイト）の「メモリ管理テーブル」が存在します．

メモリ管理テーブルには，その前後に確保されたメモリ・ブロックへのリンク・ポインタ，そのメモリ・ブロックを確保したプロセスのメモリ管理ポインタへのポインタ．そのメモリ・ブロックのエンド・アドレス＋1などの情報が収められており，Humanのメモリ管理に利用されます．ユーザーが書き換えることはできません．

*:注

あまり大きくないワークが必要とされ，かつその大きさがだいたいわかっている場合は，OSにメモリを確保してもらうよりは，プロセスの中にあらかじめワークを作っておくのが普通です．

プロセスを起動する場合，$FF4B exec ファンクションを使う限り，子プロセス実行前のメモリの割り当て，子プロセス終了後のメモリの開放は自動的に行なわれるので，ユーザーは特に意識する必要はありません．ただし，子プロセス起動の前には別の注意が必要ですので，$FF48 malloc の説明を参照してください．

□ 2.プロセス管理の概要

プロセスもメモリ上にあるのですから，当然Humanによるメモリ・ブロックの管理の結果として存在しています．そのため，プロセスの存在するメモリ・ブロックの先頭にはメモリ管理テーブルがあることはもちろん，プロセスの場合はさらに240バイトの「プロセス管理テーブル（PDB)」が続きます．

メモリ管理テーブル16バイト＋プロセス管理テーブル240バイト，計256バイトは必ずペアで利用されます．メモリ管理テーブルだけが存在するという場合はあっても，プロセス管理テーブルだけが単独で存在することはあり得ません．本書では，1組のメモリ管理テーブルとプロセス管理テーブルを指してPSPと呼ぶことにします．

プロセス管理テーブルには親プロセスへのリターン・アドレスなど，そのプロセスを管理するための情報が収められています．プロセス管理テーブルは原則として$FF4B execファンクションによって作成され，起動時のシステムの状態や起動する環境が記録されます．記録された情報はシステム資源（ファイル，メモリなど）を利用するファンクションコールを呼び出すときや，プロセスを終了する際に操作/参照されます．

PSPの構造を Fig.1.1.2 に示します．

PSPの内容をユーザーが書き換えてはいけません．熟練したユーザーであればここを書き換えてトリッキーなプログラムを作ることも可能ですが，充分な注意が必要です．

メモリ中に存在するプロセス1つ1つを区別するため，Humanはプロセスごとにプロセス IDを設定し，これによってプロセスの管理を行なっています．プロセスIDとしてはPSP＋$10（＝PDB）のアドレスが利用されます（例:PSPの先頭が$842C0のプロセスのIDは$842C0＋$10＝$842D0）．

□ 3.プロセスの起動

プロセスが格納されるメモリが確保され，ディスクからロードされるなどしてプロセスの内容がメモリに用意されると，次に行なわれるのがPSPの作成です（実際には多少の前後があります）．

プロセス管理テーブルには，起動時のシステムの状態について，以下のような情報が記録されます．

●親プロセスのUSPレジスタの値
●親プロセスのSSPレジスタの値
●親プロセスのSRレジスタの値
●アボートしたときのSRレジスタの値
●アボートしたときのSSPレジスタの値
●TRAP10〜TRAP14の値

これらは，いずれもシステム・ワーク/MPUのレジスタに存在する本物のデータのコピーで，プロセスが終了するときに利用されます．

Fig.1.1.2　PSP の構造

```
[PSP]
PSP：
  ［メモリ管理ポインタ］
  (PSP＋＄00).l    1つ前のメモリ管理ポインタ
  (PSP＋＄04).l    このメモリを確保したプロセスのメモリ管理ポインタ
  (PSP＋＄08).l    このメモリブロックの終わり＋1のアドレス
  (PSP＋＄0C).l    次のメモリ管理ポインタ

  ［プロセス管理ポインタ］
  (PSP＋＄10).l    プロセスに与えられた環境のアドレス
  (PSP＋＄14).l    プロセスの終了アドレス
  (PSP＊＄18).l    CTRL＋C によるアボート・アドレス(＄FFF1のコピー)
  (PSP＊＄1C).l    エラーによるアボート・アドレス(＄FFF2のコピー)
  (PSP＋＄20).l    プロセスに与えられたコマンドラインのアドレス
  (PSP＋＄24).l  ┐
  (PSP＊＄28).l  │ プロセスのファイル・ハンドル使用状況
  (PSP＋＄2C).l  ┘
  (PSP＋＄30).l    プロセスの BSS の先頭アドレス
  (PSP＋＄34).l    プロセスのヒープの先頭アドレス(BSS と同じ)
  (PSP＋＄38).l    プロセスの初期スタックアドレス(ヒープの終わり＋1)
  (PSP＋＄3C).l    親プロセスの USP の値
  (PSP＋＄40).l    親プロセスの SSP の値
  (PSP＋＄44).w    親プロセスの SR の値
  (PSP＋＄46).w    アボート時の SR の値
  (PSP＋＄48).l    アボート時の SSP の値
  (PSP＋＄4C).l    TRAP #10のベクターのコピー
  (PSP＋＄50).l    TRAP #11のベクターのコピー
  (PSP＋＄54).l    TRAP #12のベクターのコピー
  (PSP＋＄58).l    TRAP #13のベクターのコピー
  (PSP＋＄5C).l    TRAP #14のベクターのコピー
  (PSP＋＄60).l    プロセスのフラグ(0で親あり，－1で OS から起動)
  (PSP＋＄64).l    未使用
  (PSP＋＄68).l    子プロセスの PSP のアドレス
                   (子プロセスがない場合は0)
  (PSP＋＄6C).b  ┐
       ：        │ 未使用
  (PSP＋＄7F).b  ┘
  (PSP＋＄80).b  ┐
       ：        │ exec されたファイルのドライブ名(“n：”)
  (PSP＋＄81).b  ┘
  (PSP＋＄82).b  ┐
       ：        │ exec されたファイルのパス名
  (PSP＋＄??).b  ┘ (不定)
```

```
(PSP＋$C4).b  ┐
     :        │ execされたファイル名
(PSP＋$??).b  ┘ (不定)
(PSP＋$DC).b  ┐
     :        │ 未使用
(PSP＋$FF).b  ┘
```

この他にもプロセス管理テーブルには新しく作成するプロセスがHuman環境下で正しく実行されるための以下のような情報が納められます．

●環境変数エリアへのポインタ
● CTRL ＋ C で中断されたときのリターン・アドレス
●プロセスがエラーにより中断した場合のリターン
●現時点でのファイル・ハンドルの使用状況

以上は多くの場合現在のプロセスのものをそのまま引き継ぎます．

●プロセス終了時のリターン・アドレス
●プロセスに与えられたコマンドラインへのポインタ
●起動時におけるファイル・ハンドルの使用状況
●BSSの先頭アドレス
●ヒープの先頭アドレス
●初期スタック・アドレス
●プロセスの親あり/親なしフラグ
●execされたファイル関係の情報

以上は新しいプロセスのために用意された値が入ります．

PSPが作成されると，現在のプロセスのPSPには新しいプロセスのPSPの先頭アドレスが記録されます．

Humanには新しいプロセスのID(PSPのアドレス＋$10＝PDB) が通知され，新しいプロセスに処理が移ることが宣言されます．最後に，新しいプロセスのスタート・アドレスへジャンプして，プロセスの起動が完了します (Fig.1.1.3)．

□ 4.プロセスの終了

★非常駐終了の場合

ファンクションコール$FF00 exit，$FF4C exit2などがコールされてプロセスが非常駐終了する場合，PSPを参照して次のような処理が行なわれます．

●PSPの中のメモリ管理テーブルを参照し，このプロセスが確保したメモリとプロセス自体が格納されているメモリを開放する．
●ファイル・ハンドル使用状況などを参照し，このプロセスと子プロセスがオープンしたファイルをクローズする．
●起動時にシステム・ワーク/MPUのレジスタからコピーされてきたデータを再びシステム・ワーク/MPUのレジスタに戻す（プロセス中でシステム・ワークの内容を書き変えても，終了時には起動時のシステム状態に復帰できる）．
●親プロセスのPSP内の「子プロセスのPSPのアドレス」を0にし，子プロセスがなくなったことを記録する．

以上のような処理を行なった後，プロセス終了コードをシステム・ワークとD0レジスタに入れて，「プロセス終了時のリターン・アドレス」にしたがって親プロセスに戻ります（実際には多少の前後があります）．

★常駐終了の場合

$FF31 keepprで常駐終了する場合は事情が違ってきます．非常駐終了の場合と同様，PSPを参照して次のような処理が行なわれます．

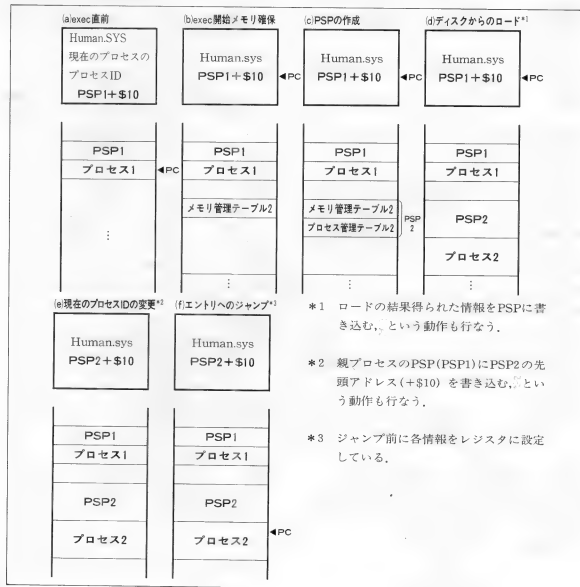
●ファイル・ハンドル使用状況などを参照し，このプロセスと子プロセスがオープンしたファイ

ルをクローズする．

- 起動時にシステム・ワーク/MPUのレジスタからコピーされてきたデータを再びシステム・ワーク/MPUのレジスタに戻す（プロセス中でシステム・ワークの内容を書き変えても，終了時には起動時のシステム状態に復帰できる）．
- 親プロセスのPSP内の「子プロセスのPSPのアドレス」を0にし，子プロセスがなくなったことを記録する．

非常駐終了とは，このプロセスで確保したメモリとプロセス自体が格納されているメモリを開放しない点で異なっています．代わりに，$FF31 keepprのコール時に指定した常駐する長さにしたがってメモリ管理テーブルの「メモリ・ブロックの最終アドレス＋1」を書き換え，メモリ・ブロックIDを$FF（常駐プロセスのメモリ）に書き換えます．

以上のような処理を行なった後，プロセス終了コードをシステム・ワークとD0レジスタに入れて，「プロセス終了時のリターン・アドレス」にしたがって親プロセスに戻ります（実際には多少の前後があります）．

Fig.1.1.3　Human の EXEC 動作

1.2 ファイル操作

Humanのファイル・システムは，ディスクフォーマットも，アクセス手順も，ほぼMS-DOSと同様な仕様となっています．

MS-DOSでは「FCBによるファイル・アクセス」と「ファイル・ハンドルによるファイル・アクセス」の2種類のファイル・アクセス方法が存在していましたが，Humanではより高度な「ファイル・ハンドルによるファイル・アクセス」のみユーザーに開放されています．

ファイル・ハンドルの概念を説明するためには，まずHuman以前のマイクロ・コンピュータ用OSのファイル操作の歴史から説明します．

□ 1.ファイル操作の歴史

先ほど，「FCB」という単語が出てきましたが，これは《ファイル・コントロール・ブロック》の略で，ファイル・アクセスに関係するパラメータ（ファイル名，ファイルの格納されているレコードナンバー，現在のファイル・アクセス・ポイントなど）を納めたテーブルを意味しています．このFCBはもとを正せばCP/M80で使われていたもので，MS-DOSはこの歴史を未だに引きずっているわけです．

CP/Mや初期のMS-DOS（バージョン1.25あたりまで）では，操作するファイルの数だけFCBを用意し，アクセスするファイルに対応するFCBのアドレスを直接指定してファイルをアクセスしていました．

この方法は「低級」であるがゆえ，ユーザーによるトリッキーなファイル・アクセスが可能であったりはしましたが，FCBを作成するのに手間がかかったり，アドレスを直接指定するのが面倒な方法でした．

そこで，MS-DOSがバージョン2.xxにバージョンアップされたときに，階層ディレクトリと共にUNIX指向の機能として導入されたのが，「ファイル・ハンドルによるファイル・アクセス」です．

「ファイル・ハンドルによるファイル・アクセス」では，OSがあらかじめいくつかのFCBを用意しておき，それに番号を付けておきます．ユーザーは，ファイル・ネーム（パスネーム）を指定するだけで，あとのFCBの作成など面倒な部分はすべてOSがやってくれます．ファイルのオープン処理を行なうと，OSはユーザーに「使うことにしたFCBの番号」を返してくれます．これが，ファイル・ハンドルです．

いったんオープンしてしまえば，ファイルを読み出すにも，書き込むにも，ユーザーはファイル・ハンドルを指定するだけでファイルにアクセスできます．

□ 2.Humanのファイル操作

このように良いことずくめのような「ファイル・ハンドル」ですが，便利であるだけに若干の制限があります．FCBを使ってファイル・アクセスをする分には，ユーザーはメモリが許す限り無制限にFCBを作成し，いくらでもファイルをオープンできます．*1　しかし，ファイル・ハンドルを使う場合は，OSが用意できるFCBが限られていることから，いちどにオープンできるファイルの数も限られてしまいます．

このいちどにオープンできるファイルの数を指定するのがCONFIG.SYSの中の"FILES＝"なのですが，MS-DOSの場合，いくら大きな数を設定しても，20以上はオープンできないという欠点がありました．さらに，OSが最初の5つのハンドルを使うので，ユーザーが使えるファイルは15個に過ぎなかったのです．

この欠点はHumanでは改善されています．8086CPUを使っていたためにアドレス空間が限られていたり，メモリがまだ高価だったりした事情から，MS-DOSの用意するファイル・ハンドル用FCBの数は一定数以上持てないようになっていましたが，X68000においてはその両方の制限事項から開放されています．したがって，もはやファイル・ハンドル用FCBを制限する必要もなく，CONFIG.SYSの"FILES＝"で指定した数だけ*2，ファイルをオープンできるようになっています．もっとも，MS-DOS同様，ファイル・ハンドルの最初の5つはOSが使っていますから，実際には「指定した数－5」です．*3

ファイルをオープンし，アクセスし，クローズするという一連の作業が1つのプロセス内で完結している場合はよいのですが，ここにプロセスの親子関係が絡んでくると，話はややこしくなります．

あるプロセスがファイルをオープンしたまま子プロセスを起動した場合，ファイル・ハンドルの状態は子プロセスにコピーされ，親プロセスがオープンしたファイルを子プロセスがそのままアクセスすることが可能です．

通常，ファイル・ハンドルはプロセスに対して5から順に与えられますが，親プロセスがにファイルをオープンしている場合は，5より大きなファイル・ハンドルから割り当てられることになります．親プロセスがオープンしたファイル・ハンドルは子プロセスでクローズしてしまうことも可能です．ただし，この場合親プロセスに戻ってもアクセスできなくなるので注意してください．

また，子プロセスがアボートするなどして，子プロセスでオープンしたファイルがクローズされずに親プロセスへ戻ってきた場合，そのファイルの面倒は親プロセスがみることになります．$FF1F allclose，$FF00 exit，$FF4C exit2などを実行した場合，子プロセスがオープンしたままのファイルもクローズされます．

OSが用意する5つのファイル・ハンドルの内容を **Fig.1.1.4** に示します．

＊1:
MS-DOS3.x では FCB でオープンできるファイル数も制限されます．これは，ネットワーク対応となり，FCB によるファイル・アクセスも OS が管理する必要が出てきたためです．

＊2:
"FILES" では5～93までを指定できます．

＊3:
OS が使っている0～4のファイル・ハンドルも，いちどクローズすることによってユーザーが使うことも可能になります．

Fig.1.1.4　OS が用意する5つの標準ハンドル

ファイルハンドル	名　称（略称）	モード	デフォルト・デバイス
0	標準入力 (stdin)	READ	CON
1	標準出力 (stdout)	WRITE	CON
2	標準エラー出力 (stderr)	WRITE	CON
3	標準外部入出力 (stdaux)	READ/WRITE	AUX
4	標準プリンタ出力 (stdprn)	WRITE	PRN

1.3 デバイス・ドライバ

MS-DOS 同様，Human ではデバイスを制御するためのデバイス・ドライバをユーザーが選択して組み込むことができ，必要があればユーザーが新たに作ることもできます．

デバイス・ドライバには2種類あり，ひとつはキャラクタ・デバイス，もうひとつはブロック・デバイスです．

キャラクタ・デバイスは1文字ずつ入出力するタイプのデバイス（例:RS-232C，プリンタなど）をサポートするためのものです．デバイス名として，それぞれに名前が指定できます（例："AUX:"，"PRN:"など）．

ブロック・デバイスはデータをブロック単位でアクセスするデバイス（例：フロッピーディスク，

RAM ディスクなど）をサポートするためのものです．デバイス名は指定できず，Human によって組み込まれた順に"A:", "B:"といった名前がつけられます．

デバイス・ドライバは，一定のフォーマットにしたがって作製されたX形式の実行型ファイルです．その先頭には，各種ベクターが収められている「デバイス・ヘッダ」が配置されています．Human とは，このデバイス・ヘッダを経由してリンクされます．その後に実際に常駐するストラテジ・ルーチン，割り込みルーチン，イニシャライズ時に1回だけ実行される非常駐部が存在する，というのが標準的なデバイス・ドライバの構成です（Fig. 1.1.5）．

Fig.1.1.5　デバイス・ドライバの構成

```
[デバイス・ドライバ]
dev header：            ＊必ずデバイス・ドライバの先頭にあること
  ─[デバイス・ヘッダ]──────────────────
  (dev_header＋$00).l      リンクポインタ
              次のデバイス・ヘッダへのポインタ．1つのデバイ
              ス・ドライバ中に2つのデバイス・ヘッダが存在する
              場合，そのアドレスを指定します．それ以外は－1
              を指定します．
  (dev_header＋$04).w      属性ワード
              デバイスの属性を示します．
              bit15   0：ブロックデバイス
                      1：キャラクタデバイス
              bit14   0：IOCTRL 不可
                      1：IOCTRL 可
              bit12
                :     常に0
              bit 6
              bit 5   0：COOKED MODE
                      1：RAW MODE
              bit 3   ＝1 CLOCK デバイス
              bit 2   ＝1 NUL デバイス
              bit 1   ＝1 標準出力デバイス
              bit 0   ＝1 標準入力デバイス
  (dev_header＋$06).l      ストラテジ・エントリポイント
              ストラテジルーチンのエントリのアドレス(strategy)
  (dev_header＋$0A).l      割り込み・エントリポイント
              割り込みルーチンのエントリのアドレス(interrupt)
  (dev_header＋$0E).b      デバイス名(8バイト)
              ブロックデバイスの場合，先頭1バイトはユニット
              数．
  (dev_header＋$15).b
stategy：             ＊必ずしもここに位置する必要はない
```

［ストラテジルーチン］

A 5にリクエスト・ヘッダが入った状態でここに飛んできます．このルーチンではA 5の内容をバッファに入れて，そのままリターンします．

interrupt：　　　　　　＊必ずしもここに位置する必要はない

［割り込みルーチン］

ストラテジ・ルーチンでバッファに納めたリクエスト・ヘッダを得て，その中のコマンド・コード従って処理を行います．

＜キャラクタ・デバイス＞	＜ブロック・デバイス＞
1 ：	1 ：ディスク交換チェック
2 ：	2 ：
3 ：IOCTRLによる入力	3 ：IOCTRLによる入力
4 ：入力	4 ：入力
5 ：先読み入力	5 ：ドライブ・コントロール&センス
6 ：入力ステータス・チェック	6 ：
7 ：入力バッファをクリア	7 ：
8 ：出力(VERIFY OFF)	8 ：出力(VERIFY OFF)
9 ：出力(VERIFY ON)	9 ：出力(VERIFY ON)
10：出力ステータス・チェック	10：
11：	11：
12：IOCTRLによる出力	12：IOCTRLによる出力

↑常駐部

↓非常駐部

0 ：初期化

この処理から帰る際にデバイスドライバの終端アドレスリクエスト・ヘッダに返すことによって，Humanはそのアドレス以降のメモリを開放して次のデバイスドライバ等に使用します。

この初期化コマンドは，初期化時に1度しか実行されませんから，通常は初期化コマンド処理ルーチンの直前のアドレスをデバイスドライバの終端アドレスとすることで常駐部のメモリを節約します。

□　1.デバイス・ドライバのコール

Humanとリンクされたデバイス・ドライバは，次のような手順でコールされます．

①リクエストヘッダへのポインタをA5に入れてストラテジ・ルーチンをコール．

A5に納められているリクエストヘッダには，デバイス・ドライバへのコマンド，パラメータが納められています．ストラテジ・ルーチンはこのアドレスをワーク（理想としてはキュー）に保存し，ひとまずリターンします．

②割り込みルーチンをコール．

割り込みルーチンは先に受け取ったリクエストヘッダの内容を解析し，それに応じたルーチンへ分岐します．

このように機能のリクエストと実際の処理を分離したのには，将来のマルチタスク化への準備という意味があります．シングルタスクである現在のバージョンではストラテジ・ルーチンと割り込みルーチンは連続してコールされるようになっています．

□ 2.デバイス・ドライバの組み込み

▼ Human 起動時の組み込み

Human起動時に組み込まれるデバイス・ドライバは，一般に”.SYS”という拡張子を持つ，コマンドラインからの実行は考えられていないXファイルで，CONFIG.SYSの”DEVICE=”にしたがって組み込まれるものです．

これらのデバイス・ドライバは，ディスクからロードされ，リロケートされると，Humanによってデバイス・ヘッダを書き換えられ，前後のデバイス・ドライバとリンクされます．

その後すぐにコマンドコード0「初期化」のリクエストヘッダを持ってストラテジ・ルーチンが呼ばれるので，初期化ルーチンはCONFIG.SYSで指定されたデバイス・ドライバへのオプション指定の解析などを行ない，デバイス・ドライバを初期化した後，デバイス・ドライバ本体の終了アドレスをリクエストヘッダに納めてrtsします．Humanは返ってきた終了アドレスまでの内容をメモリに常駐させ，その直後に次のデバイス・ドライバをロードしたり，他のワークに使ったりします．

つまり，この形式のデバイス・ドライバの場合，デバイス・ドライバ内のことだけを考えてプログラミングしていれば良いのであって，組み込みに際しての処理はすべてHumanがやってくれます．

▼ Human 起動後の追加組み込み

XCコンパイラの登場の頃から，OPMDRV.XやFLOAT?.XなどのようにHumanが起動した後から追加する形のデバイス・ドライバが登場してきました．これらのようなデバイス・ドライバは，拡張子が”.X”で，ユーザーからの指示でコマンドラインから起動された場合は自分でHumanへの組み込みを行なうようになっています．しかも，CONFIG.SYSで指定してあれば，起動時にも組み込むことができるようになっています．

CONFIG.SYSで指定しても組み込めるということは，基本的なデバイス・ドライバの形式は遵守した上で，特殊な組み込みを可能にしているということになります．そのために，このタイプのデバイス・ドライバには巧妙な仕掛けが施されています．

実は，追加型のデバイス・ドライバは，2つの初期化ルーチンを持っています．ひとつは，Humanからのコマンドコード0のリクエスト・ヘッダによる通常の初期化ルーチン．もうひとつは，コマンドラインから起動された場合に実行される初期化ルーチンです．前者についてはすでに解説したので，ここでは後者について解説します．

基本的なデバイス・ドライバの形式を遵守した場合，プログラムの先頭はデバイス・ヘッダで始まっている必要があります．普通どおりにプログラムの先頭から実行を開始するようにしていたのでは，デバイス・ヘッダの内容をコードとして実行してしまうため，まともに実行できるわけがありません．したがって，デバイス・ドライバのコーディング時にend疑似命令を使って，スタート・アドレスを変更しておく必要があります．スタート・アドレスとして指定するのは，言うまでもなく第2の初期化ルーチンの先頭アドレスです．このような仕掛けによって，コマンドラインから起動された場合，第2の初期化ルーチンに真っ先に飛んでくることになります．

第2の初期化ルーチンは次のような処理を行ないます．

①現在，最後尾のデバイス・ドライバを探し，そのデバイス・ヘッダと自分をリンクする．

②コマンドラインのアドレス（A2）から偽の初期化リクエストヘッダを作成し，本物の初期化ルーチンを直接コールする．本物の初期化ルーチンは偽のリクエスト・ヘッダにしたがってデバイス・ドライバの初期化を行ない，第2の初期化ルーチンに戻る．

③第2の初期化ルーチンは，本物の初期化ルーチンと第2の初期化ルーチンを除いたデバイス・ドライバ本体を残し，常駐終了する．

また，CONFIG.SYSで指定されて起動時に組み込まれる場合，プロセスとしてのスタート・アドレスは無視されるので，通常の組み込みが行なわれます．

1.4 Human68k Ver2.0x

Human68k Ver2.0x（以下Human2.0x)になって拡張されたのは，おもに次の6点です．

①ファイル・アクセスの高速化
②大容量メディア対応準備
③ネットワーク対応準備
④バックグラウンド処理
⑤オーバレイXファイル
⑥その他

①，②については量的な変化であるため除くとして，③～⑥に関して，それぞれ解説したいと思います．

注意!!

本書中のHuman68k ver2.0xに関する部分は，筆者の独自の解析をもとに解説されています．本資料に関するシャープ㈱への質問，問い合わせなどは一切行なわないようお願いします．本書はあくまで参考資料としてお使いください．

□ 1.ネットワーク対応準備

Human-Networkとでも呼ぶべきLANシステムの具体的な姿はHuman2.01リリース直後の現在，まだ見えていませんが，Human内部ではすでに対応が始まっています．

具体的には以下の2点です．

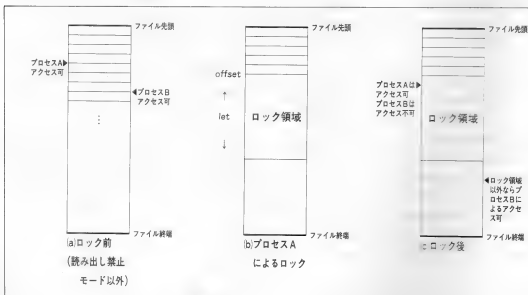
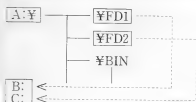
★ファイルのシェアリング（共有化）処理

LANに接続し，複数のノードが同時に1つのファイルにアクセスする場合のことを考えれば，シェアリングは重要な問題です．$FF3D openでは，ファイルのオープン時にシェアリングモードが指定できるようになりました．また，あるユーザーがファイルの内容を書き変えている間，書き変えを行なっている部分を他のユーザーがアクセスできないようにするロック機能をファンクションコール$FF5C で提供しています（Fig1.1.6)．

ファイルのシェアリング処理はCONFIG.SYSで"SHARE="を指定した場合に有効になります．"SHARE="の第1パラメータはシェアリング処理を行なうファイルの数，第2パラメータはシェアリング処理を行なうそれぞれのファイル中のロック領域の数を指定します．第1パラメータの数はあまり小さくせず，"FILES="で指定した数と同じにしておいた方がよいでしょう．"SHARE="を指定しなかった場合，Human1.0xとまったく同じファイル処理を行なうようになり，シェアリング処理は行なわれません．

このほか，複数のプロセスでテンポラリ・ファ

Fig.1.1.6 ファイルのロック機能

Fig.1.1.7

(a)ドライブをサブディレクトリとしてアクセス

ドライブAの¥FD1をアクセスした場合，実際にはドライブBが，¥FD2をアクセスすると実際にはドライブCがアクセスされる．

(b)サブディレクトリをドライブとしてアクセス

ドライブWをアクセスした場合，実際には，ドライブAの¥WORKがアクセスされる．ドライブWの¥SOURCEをアクセスすると，ドライブAの¥WORK¥SOURCEがアクセスされる．

イルを使うことを考えて，テンポラリ・ファイルを作成するコール$FF5Aが新設されています．また，すでに存在するファイルと同じ名前ではファイルを作成しないcreateコール$FF5Bも新設されました．

★仮想ドライブ処理

あるドライブを別のドライブのサブディレクトリとしてアクセスしたり，あるドライブのサブディレクトリを別のドライブとしてアクセスできる機能は，ハードディスクなどのユーザーのパス名入力の便宜を図るという目的の他に，LANを構築した場合のノード管理にも役立ちます．Human2.0xでは，この機能をファンクションコール$FF5Fで提供しています（Fig1.1.7）．

仮想ドライブに名前をつけるためには，現在使用可能なドライブの数よりも多くのドライブ名が必要です．そこで，使いたいドライブ名の最大値をCONFIG.SYSの"LASTDRIVE="で指定します．

ファイル先頭
プロセスA アクセス可
プロセスB アクセス可
⋮
ファイル終端

□ 2.バックグラウンド処理

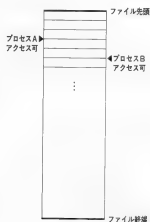
バックグラウンド処理は複数のプロセスを見かけ上同時に動かす機能です．しかし，Humanがマルチタスク OSになった，というのとは多少異なります．

なお，1989年11月現在，シャープから正式な技術資料が発表されていないため，この項では筆者が独自に解析した結果を，OS/2の概念/用語をベースとして解説しています．したがって，後に公開される正式な概念/用語とは異なっている場合があります．

スレッド

★スレッドの概念とマルチタスクの仕組み

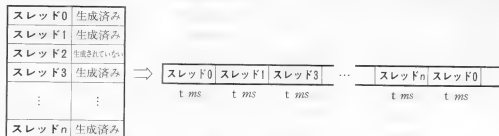
Human2.0xのバックグラウンド機能を説明する上でもっとも基本的な概念が「スレッド」です．

1つのMPUで（見かけ上）複数の処理を同時に行

なわせる場合，たった1つしかないMPUですから，ごく短い間隔（タイムスライス）ごとに複数の処理を順番に少しずつ実行させる方法が取られます．このような平行処理の手法をラウンドロビン型と呼び，Human2.0x(内のバックグラウンド・モニタ)は，Timer-Dによる割り込みを利用して実現しています．

ラウンドロビン型では，1つの処理がタイムス

Fig.1.1.8　スレッドの構造(アドレス THEAD から始まる場合)

(THREAD+$00).1	次のスレッド	
(THREAD+$04).1	現在のスレッドの状態 　$00　アクティブ 　$FE　サスペンド 　$FF　スリープ	
(THREAD+$05).b	レベルカウンタ	
(THREAD+$06).b	レベル初期値－1	
(THREAD+$07).b		
(THREAD+$08).1	PSP ID+$10(メモリ管理ポインタ)	
(THREAD+$0C).1	USP	
(THREAD+$10).1	D0	
(THREAD+$14).1	D1	
(THREAD+$18).1	D2	
(THREAD+$1C).1	D3	
(THREAD+$20).1	D4	
(THREAD+$24).1	D5	
(THREAD+$28).1	D6	
(THREAD+$2C).1	D7	
(THREAD+$30).1	A0	レジスタ保存領域
(THREAD+$34).1	A1	
(THREAD+$38).1	A2	
(THREAD+$3C).1	A3	
(THREAD+$40).1	A4	
(THREAD+$44).1	A5	
(THREAD+$48).1	A6	
(THREAD+$4C).w	SR	
(THREAD+$4E).1	PC	
(THREAD+$52).1	SSP	
(THREAD+$56).w	InDOS フラグ	
(THREAD+$58).1		
(THREAD+$5C).1	プロセス間通信バッファへのポインタ	
(THREAD+$60).b		
:	スレッド名	
(THREAD+$6F).1		
(THREAD+$70).1	スリープカウンタ	
(THREAD+$74).1	スレッドが使用できるメモリの下限	
(THREAD+$78).1	スレッドが使用できるメモリの上限	

Fig.1.1.9　スレッドの切り換え

ライスを使い切った時点で実行を中断し，別の処理を（前に中断したところから）再開します．と同時に，新たにタイムスライスのカウントダウンが始まり，次の処理切り替えタイミングまで実行を続けます．

処理を中断する際には，その処理を再開するときのために，中断した時点でのMPUやシステム・ワークの値を保存しておく必要があります．その処理を次に実行する機会が巡ってきた場合に，それらの値を元の場所に戻してやれば，中断された処理は，中断されたことをまったく意識する必要はないわけです．

保存する情報には，おもに次のようなものがあります．

- **MPUのレジスタ群**
  - D0～D7/A0～A6
  - SSP/USP (A7)
  - SR
  - PC(中断された箇所の次の命令を指している)
- **その処理のプロセスID**
- **その処理が利用できるメモリの範囲**（管理メモリ領域）

このように，1つの処理のためにMPUのレジスタやシステム・ワークの値などを保存しているものを「スレッド」と呼びます．1つのスレッドは，1つのMPUと考えることもできるでしょう．これまで，指定したタイムスライスごとに切り替えられる対象について，「処理」というあいまいな表現をしてきましたが，実際に切り替えられているのはスレッドです．

なお，スレッドは実際にはFig1.1.8のような構造をしていて，各種情報を格納しています．

スレッドの定義を行なったところで，実際に行なわれている平行処理の仕組みを解説します．

Human2.0xのバックグラウンド機能の動作環境は，CONFIG.SYSの"PROCESS＝"で設定を行ないます．

第1パラメータではスレッドの最大数(2～32)第2パラメータでは主スレッド（スレッド0，CONFIG.SYSの"SHELL＝"で指定したプログラムを実行している）のレベル(後述)，そして第3パラメータではタイムスライス（1ms～100ms）の設定を行ないます．"PROCESS＝"の指定をしなかった場合，バックグラウンド・モニタは動作せず，Timer-Dもユーザーが利用できるようになります．

スレッドにはそれぞれ0～31（CONFIG.SYSの"PROCESS＝"の第1パラメーター1）の番号がついていて，原則として番号の小さい方から順に実行され，最後のスレッドの次には最初のスレッドが実行されるリング構造になっています．最大スレッド数をn（$2 \leqq n \leqq 32$）と設定した場合のスレッド切り替えの様子をFig1.1.9に示します．

★スレッドのレベル

一般的なマルチタスク・システムでは，処理の重要度などを考慮して，処理に与えるタイムスライスを伸縮したり，処理を実行する順序を変更したりする機能があります．

Human2.0xのバックグラウンド・モニタには，"PROCESS＝"で設定されたタイムスライスを原則としてすべてのスレッドに均等に割り当てます．実行順序も，スレッド番号の小さいものから大きなものへ，という順で固定されています．しかし，

もう少しマクロな方法でスレッドの実行時間の長短を決められるようになっています.

スレッドは，それぞれ「レベル」と呼ばれる値を持っています．レベルは1～255の値をとり，値が小さいほど,MPUはそのスレッドを長時間実行することになります．この「レベル」は，実際にはスレッドが実行される頻度を示しています.

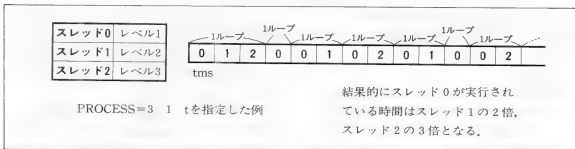
前述のとおり，2～32個のスレッドはリング構造になっており，その順番にしたがって次々に実行されて行きます．個々のスレッドはカウントダウンタイマーを持っており，このタイマーの値はそのスレッドを実行する機会が巡ってきたときに1ずつカウントダウンされ，0になったときにだけ実際にそのスレッドに処理を渡します．それ以外の場合，そのスレッドは実行せずに，次のスレッドに移ります．レベルは，このカウンタの初期値を示しています．すなわち，レベルが1のスレッドは実行する機会が巡ってくるごとに必ず実行され，レベルが2のスレッドは2回に1回，レベルが3のスレッドは3回に1回実行されることになります（Fig1.1.10).

レベルを設定できるのは，スレッドを生成するときだけで，それ以降変更できません.

Fig.1.1.10

スレッド0	レベル1
スレッド1	レベル2
スレッド2	レベル3

PROCESS＝3　1　tを指定した例

結果的にスレッド0が実行されている時間はスレッド1の2倍，スレッド2の3倍となる.

★スレッドの状態

複数のスレッドが平行処理されている場合，スレッドの状態はつねに同じではなく，いくつかの状態を変遷しつつ実行されています.

大きく分けて，スレッドの状態には次の5つがあります.

- **新規（new）**

  スレッドが新たに作成された直後の状態.

- **実行待ち（ready）**

  自分が実行される機会が巡ってくるのを待っている状態.

- **実行中（running）**

  自分が実行される機会が巡ってきて，実際に実行されている状態.

- **待機中（waiting）**

  ある条件が満たされるまで，実行を停止している状態．「ある条件」が満たされた場合，スレッドは実行待ち状態に復帰します.

  Human2.0xのバックグラウンド・モニタの場合，自発的にある時間が経過するのを待つ「スリープ」と，他のスレッドからの指令で動作を停止/再開させられる「サスペンド」の2種類の状態があります.

- **停止（halted）**

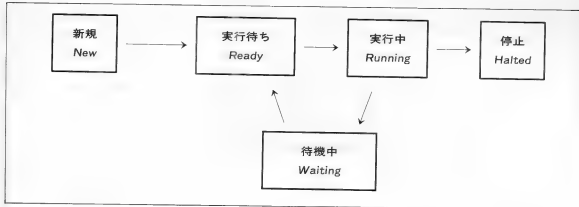
  スレッドが動作を停止している状態.

以上の状態は，Fig1.1.11のように遷移します.

★スレッドとプロセス

従来，HumanのプロセスはCOMMAND.X（あるいはVS.Xなど）を根底とし，1つの親プロセスの下に1つの子プロセスが存在し，その下に孫プロセスが存在するといった1次元的なプロセス系を形作っていました（常駐プロセスなどの例外を除く）．スレッドという概念を導入したことによって，この構造は若干変化してきます.

OS/2の場合，一般的にスレッドはプロセスの処理を分担させるために生成され，そのため『プロセスの中にスレッドが存在する』というニュアンスで説明される場合があります．また，プロセス自体も1つのスレッドによって動作しているので，スレッド同士の間にも親子関係が存在し，スレッドとプロセスでツリー状の構造を形作っていることになります．これを図にするとFig1.1.12(a)のように表現できます.

Fig.1.1.11

これに対してHuman2.0xのスレッドは親子関係を持ちません．あるスレッドと，それが生成したスレッドの間には何の関係もなく，お互いに完全に独立しています．いくつかのスレッドに処理を分担させ，それをまとめて1つの処理を行なうということもできなくはないのですが，各スレッド間に緊密な関係が存在しないため，OS/2ほどにスレッド同士が協調して働くことはあまり得意ではないようです．基本的に1つのスレッドで1つのプロセス系を実行する，と考えた方がよいでしょう．

以上のことから，Human2.0xでは『プロセスの外にスレッドが存在する』と考えることもできるでしょう．これを図にするとFig1.1.12(b)のように表現できます．

★フォアグラウンド/バックグラウンド

OS/2などでは，コンソールを占有し，ユーザーと対話できるプロセスのことを「フォアグラウンド」，そうでないプロセスを「バックグラウンド」と呼んでいます．

Human2.0xのバックグラウンド機能の場合，「バックグラウンド」という名前はついていますが，前述のような意味ではフォアグラウンドとバックグラウンドの区別は明確ではありません．

強いて言うならば，スレッド0がフォアグラウンド，それ以外がバックグラウンドであると判断し，プログラマー自身が意識してコンソールへのアクセスを管理する必要があるでしょう．

以降，スレッド0以外で実行されるプロセス系をバックグラウンドプロセスと呼ぶことにします．

★システム資源（リソース）の管理

システム資源（リソース）の管理は必要最小限のものしか行なわれていません．したがって，デバイスの利用などには注意が必要です．

ディスクなどのブロック・デバイス関しては，シェアリング処理のもとでアクセスする分には問題ありませんが，コンソールやプリンタなどのキャラクタ・デバイスへのアクセスには注意が必要です．プロセス同士で入力や出力を奪いあわないように，ユーザーが管理する必要があります．

メモリに関しては，個々のスレッド専用の管理メモリ領域を設定し，他のスレッドと競合しないようにできます．

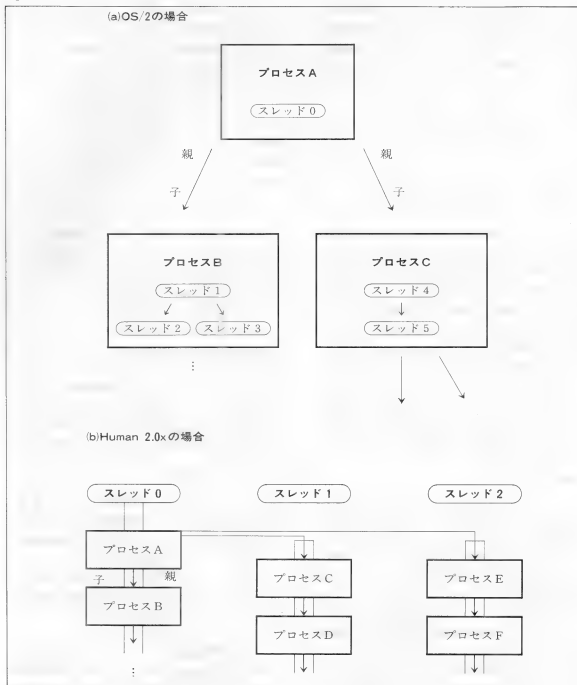
グラフィック関係のハードウェア，音源関係のハードウェア，テキストVRAM#2，#3など，もともとHuman自体の管理が甘い資源については，これまで以上に注意してユーザーが管理する必要があります．

結論として，基本的にシステム資源へのアクセスはフォアグラウンドプロセスに最優先の権利があるものと考えたほうがよさそうです．

プロセス間通信

平行処理につきものなのがプロセス間通信です（見かけ上）．同時に動いているプロセス同士が協調して動くためには，お互いにタイミングをとりあったり，データを送りあったりしなければなりません．

Human2.0xのバックグラウンド機能では，「メッセージ・システム」と「コモンエリア」の2つのプロセス間通信方法を用意しています．

Fig.1.1.12　スレッド

なお，正確には「スレッド間通信」と呼ぶべきかもしれませんが，ここでは便宜上「プロセス間通信」と呼ぶことにします．

★メッセージ・システムによる通信

送信先のスレッドのプロセス間通信バッファに直接データを送り込むのがメッセージ・システムです．これは，言ってみればプロセス間の電子メイルのようなものです．

メッセージの送信はファンクションコール$FFFD messageを使って行ないます．このコールを使えば，相手のスレッド番号を指定して，メッセージ・

データを指すポインタをはじめ，いくつかのパラメータを指定してやるだけで，あとはHumanが配達してくれます．

メッセージを受ける「郵便受け」は，プロセス間通信バッファと呼ばれ，次のような構造をしています．

```
BUFF:
        dc. l    LEN      *  0：メッセージ本体が入るバッファの大きさ
        dc. l    BODY     *  4：メッセージ本体が入るバッファを指すポインタ
        dc. w    0        *  8：メッセージアトリビュートが入る
        dc. w    $ffff    * 10：メッセージ発信側スレッド番号が入る
                 :
BODY:
        ds. b    LEN
```

他のスレッドからメッセージが送られてきたことは，プロセス間通信バッファの10バイト目が$FFFFから発信側のスレッド番号（0～31）に変化したことで判断できます．同時に，メッセージの内容がBODYで示されるバッファの中に，メッセージの付加情報がプロセス間通信バッファの8バイト目に入ります．受信側がスリープ状態にあった場合，メッセージ受信とともにスリープが解除されます．

メッセージを確認したら，受信側のプロセスはプロセス間通信バッファの10バイト目を再び$FFFFに戻し，次のメッセージを受信できる状態になったことを宣言します．

★コモンエリアによる通信

メッセージが電子メイルならば，コモンエリアは電子掲示板に例えることができます．

コモンエリアとはCONFIG.SYSの"COMMON＝"によって1KB単位で指定した広さを持つ，システム全体に共通な(COMMON)共有記憶領域のことで，ここを利用することでプロセスは他のプロセスへ，メッセージ・システムでは送りきれないような長いデータなどもやりとりできるようになります．

この機能はファンクションコール$FF55で提供されます．

コモンエリア内でやりとりされる情報はデータ・ブロックという単位でやりとりされます．データ・ブロックにはいくつかの付加情報があり，これによって管理されています．

・ブロック名

12バイト以内でデータ・ブロックの名前を示す文字列です．このブロック名でアクセスするデータを指定するので，データを送る側も受け取る側も同一のブロック名を知っている必要があります．

逆に，データ・ブロックの名前さえ知っていれば，どのプロセスもこの情報にアクセスできることになります．

・ブロック・サイズ

データ・ブロックのデータ領域のバイト数を示します．

・ロックを行なったプロセスのID，
ロック offset，length

データ・ブロックのデータ・エリア中の，他のプロセスからのアクセスをロックする領域情報，ロックを行なったプロセスのID（PSPのアドレス）が格納されています．

ファイル同様，複数のプロセスから同時にアクセスされた場合のシェアリング処理を行なっています．

なお，コモンエリアについてはバックグラウンド処理だけのものというわけではなく，一般のプロセスからも利用できます．

バックグラウンドプロセスのプログラミング―――

バックグラウンドプロセスは，厳密には「プロセス系」であり，親子関係を持った一連のプロセス群の中のひとつであると言えます．つまり，スレッド0以外で動作しているプロセスも子プロセスを起動することが可能である，ということです．もちろん，その子プロセスは親プロセスを動作させていたスレッドによって動作することになります．Fig1.1.12(b)に図示された子プロセス（図中のプロセスDやプロセスF）はバックグラウンド

プロセスであることを意識して作られたプロセスである必要はありません（ただし，資源（リソース）の共有で問題がない場合に限る）．極端な話，従来から作ってきた普通のプログラムを起動し，終了して構わないのです．しかし，そのスレッドを生成した際に，最も根底にあったプロセス（図中のプロセスCやプロセスE）は，特殊な作法でプログラミングされている必要があります．

以降，特にことわりがない限り，このようなスレッド0以外によって動作するプロセス系の最も根底にあるものをバックグラウンドプロセスと呼ぶことにします．

Human2.0xには，バックグラウンドプロセスのサンプルとして"TIMER.X"が付属してきます．おそらくこれがバックグラウンドプロセスのスタンダードな形であろうと思われるので，このプログラムを解析した結果をもとに，解説を進めたいと思います．

なお，P.473のバックグラウンド機能のサンプル・プログラムも併せて参考にしてください．

★バックグラウンドプロセスの構造

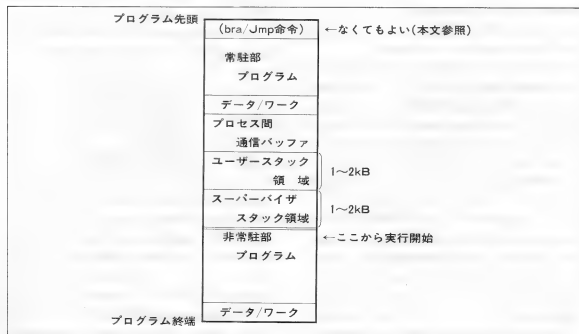
バックグラウンドプロセスは，基本的に「プロセス」です．したがって，Humanからわけてもらったメモリの中に存在し，PSPも持っているといったように，プロセスの体裁をしている必要があります．そのため，バックグラウンドプロセスも普通のプロセス同様，$FF4B execで起動されるのが普通です．

こうして起動されるバックグラウンドプロセスは，その後スレッドに登録され，メモリに常駐しなければなりません．そのため，バックグラウンドプロセスはTSR（Terminate with Stay Regident）プログラム（＝常駐プログラム）と同様な構造を持っている必要があります．すなわち，非常駐部と常駐部の2部構成という形です（Fig1.1.13）．TSRプログラムやバックグラウンドプロセスの特徴として，次の2点を押さえておいてください．

①一般的に非常駐部は常駐部の後ろに位置する．

Fig1.1.13のような形にする場合は，常駐部はすべてテキストセクションに書かなければなりません．なぜなら，データ・セクションやブロックストレージセクションを使った場合，再配置が行なわれてしまい，常駐すべきデータやワークが非常駐部の後ろに位置してしまうためです．

しかし，常駐部にはユーザースタック領域やシステム・スタック領域など，大量のワークが必要なのに，これをテキストセクションの中に置くとds疑似命令で確保したワークが圧縮されず，実行

Fig.1.1.13

ファイルが大きくなってしまいますし，データはデータ，ワークはブロックストレージ，スタックはスタックのセクションに置かないと，どうも気持ちがよくありません．これがどうしても嫌な場合は，オーバレイXファイルを使うことが考えられます．バックグラウンド機能のサンプルではこの方法を使ってみました．

②起動した際には非常駐部から実行を始める．

そのためにプログラムの先頭にbra（jmp）命令を置く，あるいは".end"疑似命令の後ろで実行開始アドレスを指定するなどの対策が必要．

●非常駐部のプログラミング

非常駐部は原則としてスレッド0で実行されます．そのため，通常のプログラムをコーディングする場合とあまり変わりません．

非常駐部はFig1.1.14に示したようなフローチャートにしたがってコーディングするとよいでしょう．この中で重要なポイントについて解説します．

スレッドの存在チェック

既に常駐部と同じプログラムがスレッドに登録されているかどうかをチェックし，存在しなければ新しいスレッドを生成して常駐部を登録し，そのスレッドをそれ以降の操作対象とします．また，既に存在している場合は，そのスレッドを以降の操作対象とします．

このチェックは，これから生成すべきスレッドの名前の入ったバッファを指定して$FFFA getthreadを呼ぶことによって行ないます(その際，スレッド番号として－1を指定する)．同名の，つまり同じプログラムを動作させているスレッドが存在している場合，そのスレッド番号が返され，存在しない場合には－1が返ってきます．

スレッドの生成

スレッドの存在チェックの結果，その必要があると判断できた場合，$FFF8 newthreadを使って新しいスレッドを生成します．その際，8つのパラメータを指定する必要があるわけですが，それぞれ次のような値を指定します．

NAME　スレッド名．存在チェックで使った名前と同じものを指定．

LVL　1～255の範囲で適当に．

BUSP　常駐部に用意したユーザー・スタック領域の最上位アドレスを指定．

BSSP　常駐部に用意したシステム・スタック領域の最上位アドレスを指定．

BSR　常駐部を実行開始するにあたってのSRの値．普通はSRの値で構わないと思われます．

ENTRY　常駐部の実行開始アドレス

BUFF　常駐部に用意したプロセス間通信バッファのアドレスを指定．

SLEEP　0を指定して，メッセージを送るまで無期限のスリープ状態にしておいた方がよいでしょう．

このようなパラメータを指定して$FFF8 newthreadを呼んだ結果，返り値として生成したスレッドの番号が得られます（エラーの場合は負の数）．

管理メモリ領域の設定

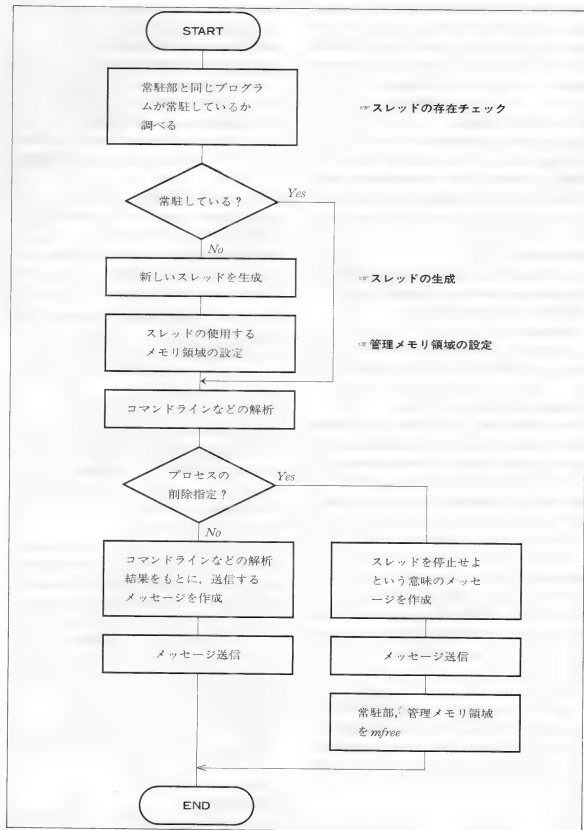
スレッドごとにプロセス系が存在することは前述のとおりですが，そのプロセス系ごとに，使うメモリ領域を設定できます．従来のHuman1.0xの場合やHuman2.0xでスレッド0だけが動いている状態では，execでプロセスをロードしたり，mallocでメモリを確保したりする場合，メイン・メモリ全域を使うことができました．

スレッドが複数存在する場合，それら複数のスレッドでexecやmallocやmfreeなどを何度も使う必要がある場合，半端なサイズのメモリ領域が多数発生し，極端にメモリ割り当ての効率が落ちる場合があります．このような場合は，あらかじめスレッドごとに管理できるメモリ領域を定めておき，その中でのみmallocなどを行なうようにすれば問題は解決できます．このような，スレッドごとに設定される管理されるメモリの領域を管理メモリ領域と呼ぶことにします．

Human2.0xにおいて管理メモリ領域を作成/設定する手順は以下のとおりです．

①malloc(malloc2)を使って，まとまったサイズの領域を得る．

②管理メモリ領域を設定するスレッドの番号，ma

Fig.1.1.14　非常駐部のプログラミング

lloc(malloc2)で得られたメモリ管理テーブル＋$10のアドレス,管理メモリ領域全体のバイト数などを指定して$FF7F setmareaをコールする.

メッセージ送信

コマンドラインで指定されたオプションを解析した結果等から常駐部の動作を決定し，それをメッセージという形で常駐部（を動かしているスレッド）に送信します．新しく生成したスレッドは，このメッセージを合図に動作を開始します．

メッセージは，あらかじめ決まった長さのデータ列（メッセージ本体）と付加情報1ワードで構成されます．TIMER.Xを見る限り，付加情報の1ワードで機能番号を，メッセージ本体でその機能についてのパラメータを表現しているようです．メッセージの送信は$FFFD messageを使って行ないます．

スレッドの停止/プロセスの削除

バックグラウンドプロセスである以上，常駐部を常駐させてスレッドに追加するのは当然ですが，必要なくなったら自身を削除する機能を備えておくのが理想的です．

スレッドを停止するのは$FFF9 killによって可能ですが，これは自分自身（現在のスレッド）を停止する機能しかなく，スレッド番号を指定して停止させることはできません．したがって，スレッドの停止は非常駐部ではなく，常駐部に自ら行なわせなければなりません．非常駐部にできることは，あらかじめ決めておいた『スレッドを停止せよ』という機能コードを持ったメッセージを送信することだけです．

なんらかの方法でスレッドが実際に停止されたことを確認したら，常駐していた常駐部と，あればば管理メモリ領域をmfreeしてメモリから開放します．

非常駐部を終了

既に常駐部が常駐していた場合，非常駐部は$FF00 exit,$FF4B exit2で非常駐終了して構いません．

常駐部が常駐しておらず，スレッドを新たに生成した場合は，$FF31 keepprを使って常駐終了します．常駐させる範囲はプログラムの先頭から常駐部の終わりまで．ここで非常駐部を後ろに持ってきた意味が出てくるわけです．

●常駐部のプログラミング

常駐部はスレッド0以外で実行されます．スレッドによって動作するプロセス系の根底にあるため，いくつかの約束を守ってコーディングする必要があります．

常駐部はFig1.1.15に示したようなフローチャートにしたがってコーディングするとよいでしょう．

この中で重要なポイントについて次に解説します．

メッセージの到着チェック

プロセス間通信バッファの10バイト目が$FFFFから送信もとのスレッド番号に変化したことでメッセージの到着を知ることができる，ということはP.23で説明しました.ここには$FFFFでなければ0～31の値が入っていることに決まっているので，実際にはtst命令でマイナスか否かをチェックするだけでもよいでしょう．

メッセージの到着チェックは，できるだけ頻繁に行なうようにしてください．Fig1.1.15のように，常駐部全体のループの中で必ず1回はチェックするようにしておくとよいと思います．

メッセージの到着を確認したら，付加情報で示された機能コードにしたがって分岐し，各処理を行ないます．

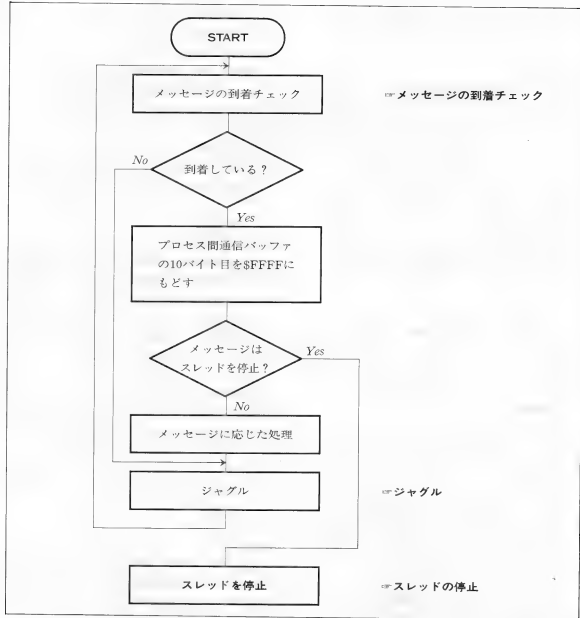
ジャグル

メッセージが届いていなかった場合など，自分のタイムスライスを消化する前に他のプロセスにMPUをあけわたした方が効率がよいと考えられる場合，$FFFF juggleをコールします．

$FFFF juggleを呼んだ場合，一見なにも起こっていないように見えますが，$FFFFを呼んで，その次の命令の実行を始めるまでの間に，スレッド群がひと回り実行されています．

スレッドの停止

スレッド0の非常駐部からのメッセージが『スレッドを停止せよ』という意味だった場合，$FFF9 killをコールして自分自身（現在のスレッド）を停止します．

Fig.1.1.15　常駐部分のプログラミング

ここで間違って$FF00 exit, $FF4B exit2などを使ってしまった場合，暴走の可能性があります．必ず$FFF9 killを使ってください．

ファンクションコール/IOCS使用上の注意

バックグラウンドプロセスでファンクションコール/IOCSを使う場合，いくつかの制限が存在します．

まず，前にも述べたとおり，システム資源（リソース）の管理が甘い関係で，標準デバイスを含めたキャラクタ・デバイスへの入出力関係では注意が必要です．暴走などの異常な動作を引き起こすまでには至らないかもしれませんが，混乱を招くことは必至です．どうしても画面表示を行ないたい場合は，IOCS$2FB PUTMESを使います．

次に，スレッド0以外で動作しているプロセス系の根底にあるプロセスでは，$FF00 exit，$FF4B exit2，$FF31 keepprを使ってはいけません．これは，TSRで同様なファンクションコールが使えないのと同じ理由で，戻るべき親プロセスとの関係

が，常駐終了した時点で失われてしまったからです．

□ 3.オーバレイXファイル

限られたメモリを有効に使うために，プログラムをいくつかの部分に分割し，プログラム全部をメモリにロードせず，必要になったとき必要な部分だけをロードして実行することをオーバレイと呼びます．これは，むしろメモリの少ない昔の機種で行なわれてきた手法で，X68Kのように豊富なメモリが実装できる機械にはあまり縁のないものと思われてきました．メモリが広く取れるようになったぶん，プログラムも高機能化/巨大化したということでしょうか．

Human2.0xで追加されたオーバレイXファイルは，Xファイルのフォーマットを一部拡張して，1つのXファイルの中にいくつものXファイルを含むような形式になっています．オーバレイXファイルの中に含まれたXファイル(オーバーレイ・モジュール，または単にモジュールと呼んでいるようです）には，それぞれモジュール番号がつけられ，$FF4B execにおいてファイル名と同時にモジュール番号を指定することでオーバレイXファイル中からのロード/実行が可能になっています．

オーバレイXファイルを起動した場合，最初メモリにはモジュール番号0のモジュールだけがロードされます．他のモジュールの機能が必要になるまでは，メモリの消費はモジュール0の領域だけに抑えることができるわけです．他のモジュールの機能が必要になった場合にだけ$FF4B execでモジュールをロードし，不必要になったモジュールはメモリから開放することで，メモリを効率よく利用できます．

オーバレイXファイルは，MS-DOSのLINK.EXEで作成されるような高水準言語向けのものではありません．メモリの管理，モジュールのローディングやその呼び出しなどは，すべてユーザーが責任を持って行なわなければいけないかわりに，アセンブラからでも手軽に利用できるという利点があります．

オーバレイXファイルの作成には，Human2.0xに標準で添付されているBIND.Xを使います．

オーバレイXファイルのサンプルとしては，バックグラウンド機能のサンプルを参照してください．このプログラムでは，メモリとディスク容量を効率よく使うためにオーバレイXファイルを使ってみました．

□ 4.その他の拡張

その他の拡張としては，

- ・ブレークモードの拡張（$FF33 breakck)
- ・メモリ割り当て方法の拡張（$FF7D)
- ・FCBの取得（$FF7C)
- ・InDOSフラグの取得（$FFF5)
- ・CONFIG.SYSファイルのコマンドの拡張

などがあります．

2章 ファンクション・コール

Human68k には OS の機能をアプリケーションに提供するためにファンクション・コールが用意されています．この章ではファンクション・コールを利用するために必要な知識と，個々のファンクションの使用法を詳しく説明します．

2.1 ファンクションコールの呼び出し方

DOSファンクションを呼び出す場合は一般的には以下の手順に従います．

①各ファンクションに渡すための値をユーザー・スタックに積む．
②未定義命令$FFに続くファンクション番号(1バイト）で構成されるファンクション・コール命令を実行する．
③ファンクション・コールから戻ってきたらユーザー・スタック・ポインタを補正．

スタックを使ってのパラメータ渡しの手法は80系CPUのユーザーにとっては馴染みが薄いことと思われますが，特に注意してほしいのは③のスタック・ポインタの補正です．うっかりしていると，忘れてしまうので慣れないうちは意識してプログラミングするようにしてください．

ファンクション・コールを使う上で以下の事項に注意してください．

①レジスタはD0以外すべて保存されます．D0を破壊されたくない場合はユーザーが保存する必要があります．
②COMMAND.XのBREAK命令などでブレイク・フラグがONになっている場合はすべてのファンクションを呼び出したときにブレイク・キー（コントロールC)がチェックされます．OFFになっている場合も，各ファンクションの解説文中で「ブレークチェックを行ないます」と記されているものについてはチェックが行なわれます．
③返り値としてD0に負の数のエラーコードが返ってくることがあります．エラーコードに関してはP.493の「ファンクション・コール・エラーコード一覧表」を参照してください．
④ファンクション・コールを簡略化するために，マクロを定義しておくと便利です．これは，その一例です．

```
FNC     macro   number
        dc. b   $ff
        dc. b   number
        endm
```

また，Cコンパイラのユーザの方は，ディレクトリ"¥INCLUDE"の中のDOSCALL.MACをアセンブラ・ソースの先頭でインクルードしておくことで，"DOS _PRINT"のようにファンクション・コールを行なうことができます．

2.2 個々のファンクション・コール使用法解説

各ファンクション・コールの解説は原則として番号順にしていきます．

各ファンクションについてサンプル・プログラムを紹介し，実行結果も示しています．サンプル・プログラムはアセンブラで記述されていますので，以下の手順で実行ファイルを作成し，ファンクションの動作を確認してください．

①"ed.x" でソース・ファイルを作成
②"as.x" でアセンブル
③"lk.x" でリンク

一連のサンプル・プログラムでは，よく使うマクロを外部ファイル("MACRO.H")にして，プログラムの先頭でincludeして使っています．あらかじめ"MACRO.H"をディスク上に作成し，インクルードできるようにしておいてください．

```
*
*       macro defination
*
FNC             macro   number
                dc.b    $ff,number
                endm
IOCS            macro   number
                move.l  #number,d0
                trap    #15
                endm
```

```
PUSH            macro   reg_list
                movem.l reg_list,-(sp)
                endm
POP             macro   reg_list
                movem.l (sp)+,reg_list
                endm

TRUE            =       -1
FALSE           =       0
```

リスト：MACRO.H

解説の例

$FF02 putchar(CODE) 1.0x/2.0x

(ファンクションコール命令) (ファンクション名) (引数リスト) (Human のバージョン)

引数(引数の名前と型)

word　CODE　(ただし$0000～$00FF)

返り値(返り値の入るレジスタやバッファとその内容)

なし.

機能(機能の概略)

指定した文字コードを標準出力（テキスト画面）に出力します.

参照に示した関連ファンクション・コール$FF06 inpout(CODE), $FF1D fputc(CODE, FILENO)（出力ファイルとして"con"を指定した場合）にも共通して言えることですが，これらのファンクション・コールで画面にシフトJISコードを１バイトずつ出力した場合も，漢字は正しく表示されます.

ブレイクチェックを行ないます（^C,^S,^P,^N).

コール　(一般的なコール方法)

```
        move.w  #CODE,-(SP)
        dc.w    $FF02
        addq.l  #2,SP
```

参照（関連するファンクションコール）

$FF04　comout(CODE)
$FF05　prnout(CODE)
$FF06　inpout(CODE)
$FF09　print(MESPTR)
$FF10　consns()
$FF23　conctrl(MD,??)

サンプル・プログラム（サンプル・プログラムのリスト，実行結果など）

P.198　SMPL1

仮マークの表示があるファンクションコールは，筆者が独自に解析したものです.

$FF00 exit[ ]　　1.0x/2.0x

引数

なし.

返り値

なし（戻ってこない）.

機能

ユーザー・プログラムを非常駐終了します．親プロセスへは終了コードとして 0 が返されます．その他の値を返したい場合は$FF4C exit2を使います.

現在オープンされているハンドルは子プロセスがオープンしたハンドル含めてすべてクローズされます.

コール

```
        dc.w    $FF00
```

参照

$FF31　keeppr(PRGLEN, CODE)

$FF4C　exit2(CODE)

サンプル・プログラム

P.198　SMPL1

$FF01 getchar[ ]　　1.0x/2.0x

引数

なし.

返り値

D0.L　キーコード（ただし$00000000～$000000FF）

機能

キーボードから 1 文字入力します．入力がない場合は入力されるまで待ちます．押されたキーは画面にエコーバックされます.

ブレークチェックを行ないます（^C,^P,^N）.

同様な機能を持つコールがいくつかありますが，それぞれ微妙に異なっています．下の表を参考にして，適当なコールを選択して使ってください.

コール	入力待ち	エコーバック	ブレークチェック
$FF01 getchart	○	○	○
$FF06 inpout ($FF)	×	×	×
$FF07 inkey	○	×	×
$FF08 getc	○	×	○

コール

```
        dc.w    $FF01
```

参照

$FF03　cominp()

$FF06　inpout(CODE)

$FF07　inkey()

$FF08　getc()

$FF0A　gets(INPPTR)

$FF0B　keysns()

$FF0C　kflush(MODE, ??)

$FF1A　getss(INPPTR)

$FF24　keyctrl(MD, ??)

サンプル・プログラム

P.198　SMPL1

$FF02 putchar(CODE)　1.0x/2.0x

引数

word　CODE　(ただし$0000～$00FF)

返り値

なし.

機能

指定した文字コードを標準出力(テキスト画面)に出力します.

参照に示した関連ファンクション・コール$FF06 inpout(CODE), $FF1D fputc(CODE, FILENO)(出力ファイルとして"con"を指定した場合)にも共通して言えることですが, これらのファンクション・コールで画面にシフトJISコードを1バイトずつ出力した場合も, 漢字は正しく表示されます.

ブレイクチェックを行ないます(^C, ^S, ^P, ^N). ブレイクチェックが不用な場合は, $FF06 inpoutを使います.

コール

```
move.w  #CODE,-(SP)
dc.w    $FF02
addq.l  #2,SP
```

参照

$FF04　comout(CODE)

$FF05　prnout(CODE)

$FF06　inpout(CODE)

$FF09　print(MESPTR)

$FF10　consns()

$FF1D　fputc(CODE, FILENO)

$FF1E　fputs(BUFFER, FILENO)

$FF23　conctrl(MD, ??)

サンプル・プログラム

P.198　SMPL1

$FF03 cominp( )　　1.0x/2.0x

引数

なし.

返り値

D0.L　回線からのデータ（ただし$00000000～$000000FF).

機能　2

RS-232C回線から1バイト入力します.

入力がない場合は入力があるまで待ちます．待たれては困る場合は，あらかじめ$FF12 cinsns()，IOCSの$33 ISNS232Cなどで入力データがあるかどうかチェックするようにしてください.

ブレイクチェックを行ないます（^C).

コール

```
        dc.w    $FF03
```

参照

$FF01　getchar()

$FF12　cinsns()

サンプル・プログラム

P.199　SMPL2

$FF04 comout(CODE)　　1.0x/2.0x

引数

word　CODE　（ただし$0000～$00FF)

返り値

なし.

機能

RS-232C回線へ1バイト出力します.

Xコントロールがonの設定で，Xoffコード（$13）を受信している場合，出力は行なわず，Xonコード（$11）を受信するのを待ちます．待たれては困る場合は$FF13 coutsns()で出力できるかどうかをチェックするようにしてください.

ブレイクチェックを行ないます（^C).

コール

```
        move.w  #CODE,-(SP)
        dc.w    $FF04
        addq.l  #2,SP
```

参照

$FF02　putchar(CODE)

$FF13　coutsns()

サンプル・プログラム

P.199　SMPL2

$FF05 prnout[CODE]　　1.0x/2.0x

引数

word　CODE　(ただし$0000～$00FF)

返り値

D0.L　－1

機能

プリンタへ1文字出力します．CONFIG.SYSで組み込んだ"prn" デバイス・ドライバを通して出力するので，漢字コードを処理する必要のない場合はIOCSの$3E OUTLPTを使うか，デバイス"lpt" をオープンし，そのハンドルへ出力するようにします．

プリンタが印字できる状態にない場合は，印字できるようになるまで待ちます．待たれては困る場合は$FF11 prnsns()で出力できるかどうかチェックしてください．

ブレイクチェックを行ないます (^C)．

コール

```
move.w  #CODE, -(SP)
dc.w    $FF05
addq.l  #2, SP
```

参照

$FF11　prnsns()

サンプル・プログラム

P.200　SMPL3

キーボードオペレーション

X68000のIOCSはキーボード割り込みによってキー入力を調べています．そして，特定のキー入力に対しては割り込みルーチン内部で処理を実行します．代表的な処理としては COPY によるハードコピーがありますが，その他に次のような処理を実行できます．

キー	処理
CTRL + F1	ドライブAのイジェクト
CTRL + F2	ドライブBのイジェクト
CTRL + F3	ドライブCのイジェクト
CTRL + F4	ドライブDのイジェクト
CTRL + F5	ドライブEのイジェクト
CTRL + F7	ファンクション・キー表示欄の表示内容変更 しない・通常表示・SHIFT モードの表示
OPT. 1 + COPY	プリンタ1ページ送り
OPT. 2 + COPY	プリンタ改行
CTRL + OPT. 1 + DEL	リセット

$FF06 inpout[CODE] 1.0x/2.0x

引数

word　　CODE　(ただし$0000〜$00FF)

返り値

★ CODE が$FF または$FE の場合

D0.L　　キー・コードを返します．$00000000の場合は，キー入力がなかったことを表わします．

★ CODE が$FF,$FE 以外の場合

D0.L　　0

機能

★ CODE が$FF の場合

キー入力します(入力がない場合も待ちません)．エコーバック，ブレイクチェックは行ないません．

キー・コードはD0に返ります．

★ CODE が$FE の場合

キー・センスします．$FFとの違いは，$FFが入力されたキーのコードを次々に「受け取って」行くのに対し，$FEは入力された最初のキーのコードを「参照する」だけである点です．したがって，さらに$FF01 getchar()などのファンクションを実行するまでは，$FEの値を持ってこのファンクションを何度コールしても同じ値が返ってくることになります．

キー・コードはD0に返ります．

ブレイクチェックは行ないません．

★ CODE が$FF,$FE 以外の場合

文字コードとして画面に出力します．シフトJISコードを1バイトずつ表示させることによって，漢字も表示できます．

ブレイクチェックは行ないません．

コール

```
move.w  #CODE,-(SP)
dc.w    $FF06
addq.l  #2,SP
```

参照

$FF01　getchar()
$FF02　putchar()
$FF07　inkey()
$FF08　getc()
$FF09　print(MESPTR)
$FF0A　gets(INPPTR)
$FF0B　keysns()
$FF0C　kflush(MODE,PTR)
$FF10　consns()
$FF1A　getss(INPPTR)
$FF23　conctrl(MD,??)

サンプル・プログラム

P.199　SMPL2

$FF07 inkey( )　　1.0x/2.0x

引数

なし.

返り値

D0.L　　キー・コード（ただし$00000000～$000000FF).

機能

キーが押されるまで待ち，押されたキー・コードを返します．押されたキーは画面にエコーバックされません.

ブレイクチェックは行ないません.

コール

```
	dc.w	$FF07
```

参照

$FF01　getchar()
$FF06　inpout(CODE)
$FF08　getc()
$FF0A　gets(INPPTR)
$FF0B　keysns()
$FF0C　kflush(MODE, PAR)
$FF23　conctrl(MD, ??)

サンプル・プログラム

P.198　SMPL1

$FF08 getc( )　　1.0x/2.0x

引数

なし.

返り値

D0.L　　キー・コード（ただし$00000000～$000000FF).

機能

キーが押されるまで待ち，押されたキー・コードを返します．押されたキーは画面にエコーバックされません.

ブレイクチェックを行ないます（^C,^P,^N).

コール

```
	dc.w	$FF08
```

参照

$FF01　getchar()
$FF06　inpout(CODE)
$FF07　inkey()
$FF0A　gets(INPPTR)
$FF0B　keysns()
$FF0C　kflush(MODE, PAR)
$FF23　conctrl(MD, ??)

サンプル・プログラム

P.198　SMPL1

$FF09 print(MESPTR) 1.0x/2.0x

引数

long　　MESPTR ; 文字列を指すポインタ.

返り値

なし.

機能

　NULL・コード ($00) までの文字列を表示します. コントロール・コードやエスケープシーケンスなども有効です.

　ブレイクチェックを行ないます (^C,^S,^P,^N).

コール

```
        pea     MESPTR
        dc.w    $FF09
        addq.l  #4,SP
                :
MESPTR:
        dc.b    'X68000',13,10,0
```

参照

$FF02　putchar()
$FF06　inpout(CODE)
$FF10　consns()
$FF1D　fputc(CODE,FILENO)
$FF1E　fputs(BUFFER,FILENO)
$FF23　conctrl(MD,??)

サンプル・プログラム

P.201　SMPL4

$FF0A gets[INPPTR] 1.0x/2.0x

引数

long　INPPTR　；入力バッファを指すポインタ．

返り値

.l　入力文字数．(INPPTR＋1).bに入っている値と同じ（0～255）．

(INPPTR＋0).b　入力最大文字数．あらかじめ呼ぶ側で値を入れておきます．

(INPPTR＋1).b　実際に入力された文字数が入ります（エンドマーク$00は数えません）．

(INPPTR＋2)以降　入力された文字列が入ります．入力最大文字数＋1バイトのエリアを確保しておく必要があります（エンドマーク$00が入るため）．

機能

CRコードまで文字列を入力し，入力バッファに格納します．最大入力文字数を超えたときは警告を出し，終了しません．入力された文字列の末尾にはCRコードの代わりにエンドマーク$00が付加されます．押されたキーは画面にエコー・バックされます．

文字列入力中にはBSなどの編集キーや，テンプレート機能などが使えます．Human2.0xでヒストリ・ドライバを組み込んでいる場合は，ヒストリ機能，エイリアス機能なども使えます．

ブレイクチェックも行ないます（^C，^P，^N）．

呼び出す際はあらかじめ(INPPTR＋0).bに最大入力文字数を書き込んでおくことを忘れないようにしてください．

コール

```
        pea     INPPTR
        dc.w    $FF0A
        addq.l  #4,SP
                :
INPPTR:
        dc.b    40
        dc.b    0
        ds.b    41
```

参照

$FF01　getchar()
$FF06　inpout(CODE)
$FF07　inkey()
$FF08　getc()
$FF0B　keysns()
$FF0C　kflush(MODE,??)
$FF1A　getss(INPPTR)
$FF24　keyctrl(MD,??)

サンプル・プログラム

P.201　SMPL4

$FF0B keysns( )　1.0x/2.0x

引数

なし．

返り値

D0.L　キー入力状態　0:入力なし
　　　　　　　　　　−1:入力有り

機能

キー入力状態をチェックします．先行入力バッファにデータが入っているかどうかをチェックするので，現在のキーの状態とは異なる場合があります．

ブレイクチェックを行ないます（^C,^P,^N）．

コール

```
        dc.w    $FF0B
```

参照

$FF01　getchar()
$FF06　inpout(CODE)
$FF07　inkey()
$FF08　getc()
$FF0A　gets(INPPTR)
$FF0B　keysns()
$FF0C　kflush(MODE,??)
$FF1A　getss(INPPTR)
$FF24　keyctrl(MD,??)

サンプル・プログラム

P.202　SMPL5

$FF0C kflush(MODE,???)　1.0x/2.0x

引数

word　MODE　；このファンクションの動作モードを指定します．$01,$06,$07,$08,$0A，そして，それ以外の値をとり，それぞれ異なった動作をします．その他のパラメータは，指定した動作モードによって意味や型が違ってきます．

返り値

★ MODEが$01の場合

D0.L　キー・コード（ただし$00000000〜$000000FF）

★ MODEが$06の場合

D0.L　キー・コードを返します．$00000000の場合は，キー入力がなかったことを表わします．

★ MODEが$07の場合

D0.L　キー・コード（ただし$00000000〜$000000FF）．

★ MODE が$08 の場合

●PARが$FE, $FFの場合

D0.L　　キー・コード（ただし$00000000～$000000FF）.

●PARがその他の値の場合.

なし.

★ MODE が$0A の場合

D0.L　入力文字数.（INPPTR＋1).bに入っている値と同じ（0～255).

(INPPTR＋0).b　入力最大文字数. あらかじめ呼ぶ側で値を入れておきます.

(INPPTR＋1).b　実際に入力された文字数が入ります（エンドマーク$00は数えません）.

(INPPTR＋2)以降　入力された文字列が入ります. 入力最大文字数＋1バイトのエリアを確保しておく必要があります（エンドマーク$00が入るため）.

★ MODE がその他の値の場合

なし.

機能

キーバッファをクリアしてからMODEで示される番号のファンクション・コールに相当する動作を行ないます. MODEに$01, $06, $07, $08, $0A以外の値を指定した場合は, キーバッファをクリアするだけで他の動作は行ないません.

★ MODE が$01 の場合

kflush($01)

キーバッファをクリアして, キーボードから1文字入力します. 入力がない場合は入力されるまで待ちます. 押されたキーは画面にエコーバックされます.

ブレイクチェックを行ないます（^C,^P,^N).

★ MODE が$06 の場合

kflush($06,CODE)

word　CODE

キーバッファをクリアして, ファンクション$FF06 inpoutと同等な動作を行ないます. 動作はPARによって指定されます. $FF06 inpoutを参照してください.

★ MODE が$07 の場合

kflush($07)

キーバッファをクリアして, キーが押されるまで待ち, 押されたキーコードを返します. 押されたキーは画面にエコーバックされません.

ブレイクチェックは行ないません.

★ MODE が$08 の場合

kflush($08)

キーバッファをクリアして, キーが押されるまで待ち, 押されたキーコードを返します. 押されたキーは画面にエコーバックされません.

ブレイクチェックを行ないます（^C,^P,^N).

★ MODE が$0A の場合

kflush($0A,INPPTR)

long　INPPTR

キーバッファをクリアして, CRコードまで文字列を入力し, 入力バッファに格納します. 最大入力文字数を超えたときは警告を出し, 終了しません. 入力された文字列の末尾には

CRコードの代わりにエンドマーク$00が付加されます.押されたキーは画面にエコーバックされます.

文字列入力中にはBSなどの編集キーや，テンプレート機能が使えます.

ブレイクチェックも行ないます（^C,^P,^N).

呼び出す際はあらかじめ(INPPTR+0).bに最大入力文字数を書き込んでおくことを忘れないように.

コール

```
        pea      INPPTR
        move.w   #$0A,-(SP)
        dc.w     $FF0C
        addq.l   #6,SP
                 :
INPPTR:
        dc.b     40
        dc.b     0
        ds.b     40+1
```

参照

$FF01　getchar()
$FF06　inpout(CODE)
$FF07　inkey()
$FF08　getc()
$FF0A　gets(INPPTR)

サンプル・プログラム

P.202　SMPL5

$FF0D fflush[ ]　1.0x/2.0x

引数

なし.

返り値

なし.

機能

使用中のトラックバッファをディスクに書き出してクリアし，ディスクをリセットします.ファイルのクローズとは意味が違うので注意してください.

コール

```
        dc.w     $FF0D
```

参照

$FF0E　chgdrv(DRIVE)
$FF29　namests(FILE,BUFFER)
$FF36　dskfre(DRIVE,BUFFER)
$FF3C　creat(NAMEPTR,ATR)
$FF3D　open(NAMEPTR,MODE)
$FF3E　close(FILENO)
$FF4E　files(FILBUF,NAMPTR,ATR)
$FF4F　nfiles(FILBUF)

サンプル・プログラム

P.205　SMPL6

$FF0E chgdrv(DRIVE) 1.0x/2.0x

引数

word　DRIVE　；ドライブ番号．A=0, B=1, …

返り値

D0.L　指定可能なドライブ数を返します(１～N)．指定した値以下の数が返されたときはエラーです．

機能

カレント・ドライブを指定します．

コール

```
        move.w  #DRIVE,-(SP)
        dc.w    $FF0E
        addq.l  #2,SP
```

参照

$FF0D　fflush()
$FF18　curdrv()

サンプル・プログラム

P.206　SMPL7

*.X デバイス・ドライバ

Human68Kは各種デバイスを操作するためのプログラムをデバイス・ドライバとして登録できます．本文でも述べたように通常はCONFIG.SYSにファイルを指定しますが，最近のデバイス・ドライバの中には*.Xファイルになっていて，起動することによって登録できるものもあります．このようなプログラムは登録するタイミングが2回あることになります(CONFIG.SYSかコマンドとしてか)．

どちらのタイミングで登録しても，デバイス・ドライバの実行内容は同じです．しかし，安全性の面では差があります．と言うのは，CONFIG.SYSで登録した場合はプログラムがスーパーバイザ・エリアに置かれるのに対し，コマンドとして実行して登録した場合はプログラムはユーザー・エリアに置かれるためです．ですから，安全性が必要な場合は，なるべくCONFIG.SYSで登録した方が良いでしょう．

$FF0F drvctrl(MD * 256+DRIVE) 1.0x/2.0x

引数

word　MD*256＋DRIVE　;MDはこのファンクションの機能選択で０～５の値を取ります．DRIVEは対象とするドライブの指定で，カレント・ドライブなら０,Aドライブなら１，Ｂドライブなら２を指定します．

返り値

MD＝0の場合にのみ値が返されます．

D0.L　指定ドライブの状態．

bit7　LED点滅
bit6　イジェクト禁止
bit5　バッファ有り
bit4　ユーザーによるイジェクト禁止
bit3　PRO（プロテクト＝1)
bit2　RDV（ノットレディ＝1)
bit1　メディア挿入
bit0　誤挿入

機能

指定ドライブの状態をチェック・設定します．

★ MDが０の場合
ドライブの状態をチェックします．状態はD0に返ります．

★ MDが１の場合
イジェクトを行ないます．イジェクト前に，オープンされているファイルはクローズされ，バッファは消去されます．

★ MDが２の場合
イジェクトを禁止します．この指定を行なった後はMD＝1のイジェクトも禁止されます．

★ MDが３の場合
イジェクトを許可します(実際にイジェクトが行なわれるわけではありません)．オープンされているファイルはクローズされません．バッファは消去されます．

★ MDが４の場合
ディスクが挿入されていないときにLEDを点滅させるモードにします．

★ MDが５の場合
ディスクが挿入されていないときにLEDを消灯させるモードにします．

コール

```
move.w  #MD*256+DRIVE,-(SP)
dc.w    $FF0F
addq.l  #2,SP
```

サンプル・プログラム

P.205　SMPL6

$FF10 consns( )　　1.0x/2.0x

引数

なし.

返り値

D0.L　　画面への出力可否　　0:出力不可
　　　　　　　　　　　　　　−1:出力可能

機能

画面へ出力可能かどうかを調べます.

参照

$FF02　putchar()
$FF06　inpout(CODE)
$FF09　print(MESPTR)
$FF23　conctrl(MD,??)

サンプル・プログラム

サンプル・プログラムは特に掲載しません. 通常画面への出力が不可になるということはまずありません. 標準出力を画面からAUXなどにリダイレクトした場合を考慮するならば, 画面に出力する前にこのファンクションでチェックを行なうということも考えられます.

$FF11 prnsns( )　　1.0x/2.0x

引数

なし.

返り値

D0.L　　プリンタへの出力可否　　0:出力不可
　　　　　　　　　　　　　　　　−1:出力可能

機能

プリンタへ出力可能かどうかを調べます.

どのような場合に出力不可となるかはプリンタ・ドライバの仕様によって異なります.

コール

```
dc.w    $FF11
```

参照

$FF05　prnout(CODE)

サンプル・プログラム

P.200　SMPL3

$FF12 cinsns( )　　1.0x/2.0x

引数

なし.

返り値

D0.L　RS-232Cからの入力可否　0:入力不可
　　　　　　　　　　　　　　−1:入力可能

機能

RS-232Cから入力可能かどうかを調べます.

このファンクションは,『NULLコードを受信した場合はそれ以降《入力不可》となる』仕様になっていますので,注意してください.

コール

```
dc.w    $FF12
```

参照

$FF03　cominp()

サンプル・プログラム

P.199　SMPL2

$FF13 coutsns( )　　1.0x/2.0x

引数

なし.

返り値

D0.L　RS-232Cへの出力可否　0:出力不可
　　　　　　　　　　　　　−1:出力可能

機能

RS-232Cへ出力可能かどうかを調べます.

コール

```
dc.w    $FF13
```

参照

$FF04　comout(CODE)

サンプル・プログラム

P.199　SMPL2

$FF17 fatchk(FILE,BUFFER) 1.0x/2.0x

引数

long　FILE　；ファイル・ネームを指すポインタ．
long　BUFFER　；結果が入るバッファを指すポインタ．

返り値

D0.L　使ったバッファのバイト数（エンドマークの$0000も含む）．負の数の場合はエラーコードです．
(BUFFER＋0).w　ドライブ番号 A=1, B=2, …
(BUFFER＋2).w　最初のセクタ番号
(BUFFER＋4).w　セクタ数
(BUFFER＋6).w　次のセクタ番号
(BUFFER＋8).w　セクタ数
　：
(BUFFER＋((n＋1)*2)).w　$0000　（エンドマーク）

機能

指定したファイルの使っているセクタのチェイン情報を返します．

複雑にチェインしているファイルの場合，大きめのバッファを用意しておく必要があります．

返り値のD0の値が8の場合は，ファイルが「連続したセクタに存在している」ことがわかります．

コール

```
        pea     BUFFER
        pea     NAME
        dc.w    $FF17
        addq.l  #8,SP
                :
FILE:
        dc.b    'HUMAN.SYS',0
BUFFER:
        ds.w    1       *ドライブ番号が入る．
        ds.w    2*n     *セクタ情報が入る．
```

参照

$FF29　namests(FILE,BUFFER)
$FF4E　files(FILEBUF,NAMEPTR,ATR)
$FF4F　nfiles(FILEBUF)

サンプル・プログラム

P.207　SMPL8

$FF18 hendsp(MD,???)　　1.0x/2.0x

引数

word　MD　　; このファンクションの動作モードを指定します.
　　　　　　　その他のパラメータは, 指定したモードによって意味や型が違ってきます.

返り値

★ MD が 0 の場合
　D0.L　モード表示ウィンドウの最大文字数

★ MD が 1 の場合
　D0.L　次のPOS

★ MD が 2 の場合
　D0.L　次のPOS

★ MD が 4 の場合
　D0.L　入力ウィンドウの最大文字数

★ MD が 5 の場合
　D0.L　次のPOS

★ MD が 6 の場合
　D0.L　次のPOS

★ MD が 8 の場合
　D0.L　候補ウインドウの最大文字数

★ MD が 9 の場合
　D0.L　次のPOS

★ MD が10の場合
　D0.L　次のPOS

★ MD がその他の値の場合
なし.

機能

漢字変換行をコントロールします.

このファンクションは日本語FPから使うので, 一般のアプリケーションが使ってはいけません.

★ MD が 0 の場合
　hendsp(0)
　　モード表示ウインドウをオープンします.

★ MD が 1 の場合
　hendsp(1, POS, MESPTR)
　word　POS　　; モード表示ウインドウ中の位置
　long　MESPTR ; 表示文字列を指すポインタ
　　モード表示ウインドウの中のPOSの位置から, MESPTRで指された文字列をノーマル文字で表示します.

★ MD が 2 の場合
　hendsp(2, POS, MESPTR)
　word　POS　　; モード表示ウインドウ中の位置

long　MESPTR　; 表示文字列を指すポインタ

モード表示ウインドウの中のPOSの位置から,MESPTRで指された文字列をリバース文字で表示します.

★ MD が 3 の場合

hendsp(3)

モード表示ウインドウをクローズします.

★ MD が 4 の場合

hendsp(4)

入力ウインドウをオープンします.

★ MD が 5 の場合

hendsp(5, POS, MESPTR)

word　POS　; 入力ウインドウ中の位置

long　MESPTR　; 表示文字列を指すポインタ

入力ウィンドウの中のPOSの位置から,MESPTRで指された文字列をノーマル文字で表示します.

★ MD が 6 の場合

hendsp(6, POS, MESPTR)

word　POS　; 入力ウインドウ中の位置

long　MESPTR　; 表示文字列を指すポインタ

入力ウインドウの中のPOSの位置から,MESPTRで指された文字列をリバース文字で表示します.

★ MD が 7 の場合

hendsp(7, POS)

word　POS　; 入力ウインドウ中の位置

POS以降をもとに戻します.

★ MD が 8 の場合

hendsp(8)

候補ウインドウをオープンします.

★ MD が 9 の場合

hendsp(9, POS, MESPTR)

word　POS　; 候補ウインドウ中の位置

long　MESPTR　; 表示文字列を指すポインタ

候補ウインドウの中のPOSの位置から,MESPTRで指された文字列をノーマル文字で表示します.

★ MD が10の場合

hendsp(10, POS, MESPTR)

word　POS　; 候補ウインドウ中の位置

long　MESPTR　; 表示文字列を指すポインタ

候補ウインドウの中のPOSの位置から,MESPTRで指された文字列をリバース文字で表示します.

★ MD が11の場合

hendsp(11)

候補ウインドウをクローズします.

コール

(例はMDが５の場合)

```
		pea	MSG1
		move.w	#1,-(SP)
		move.w	#5,-(SP)
		dc.w	$FF18
		addq.l	#8,SP
			:
MSG:
		dc.b	'入力ウィンドウ',0
```

参照

$FF22　knjctrl(MD,??)

サンプル・プログラム

原則として一般のアプリケーションから使うことはないので，特にサンプル・プログラムは掲載しません．

$FF19 curdrv[ ]　　1.0x/2.0x

引数

なし．

返り値

D0.L　　ドライブ番号 A=0,B=1,…

機能

カレント・ドライブのドライブ番号を返します．返り値に$41を加えることでドライブ・ネーム（'A'～'Z'のキャラクタコード）を得ることができます．

コール

```
		dc.w	$FF19
```

参照

$FF0E　chdrv(DRIVE)

サンプル・プログラム

P.206　SMPL7

$FF1A getss(INPPTR) 1.0x/2.0x

引数

long INPPTR ；入力バッファを指すポインタ.

返り値

DO.L 入力文字数. (INPPTR＋1).bに入っている値と同じ（0～255).

(INPPTR＋0).b 入力最大文字数. あらかじめ呼ぶ側で値を入れておきます.

(INPPTR＋1).b 実際に入力された文字数が入ります（エンドマーク$00は数えません).

(INPPTR＋2)以降 入力された文字列が入ります. 入力最大文字数＋1バイトのエリアを確保しておく必要があります（エンドマーク$00が入るため).

機能

CRコードまで文字列を入力し，入力バッファに格納します. 最大入力文字数を超えたときは警告を出し，終了しません. 押されたキーは画面にエコーバックされます.

$FF0A getsとの違いは，ブレイクチェックを行なわない点です.

コール

```
        pea     INPPTR
        dc.w    $FF1A
        addq.l  #4,SP
                :
INPPTR:
        dc.b    40
        dc.b    0
        ds.b    41
```

参照

$FF01 getchar()
$FF06 inpout(CODE)
$FF07 inkey()
$FF08 getc()
$FF0A gets(INPPTR)
$FF0B keysns()
$FF0C kflush(MODE,??)
$FF24 keyctrl(MD,??)

サンプル・プログラム

P.201 SMPL4

$FF1B fgetc(FILENO)　　1.0x/2.0x

引数

word　FILENO　；ファイル・ハンドル

返り値

D0.L　読み出した文字コード（ただし$00000000～$000000FF）．ファイル終端ならば－1．

機能

ファイルから1文字読み出します．入力があるまで待ち，ブレイクチェックは行ないません．

コール

```
move.w  #FILENO,-(SP)
dc.w    $FF1B
addq.l  #2,SP
```

参照

$FF1C　fgets(BUFFER,FILENO)
$FF3D　open(NAMEPTR,MODE)
$FF3F　read(FILENO,DATAPTR,SIZE)

サンプル・プログラム

P.207　SMPL9

$FF1C fgets(INPPTR,FILENO)　　1.0x/2.0x

引数

long　INPPTR　；$FF0A getsなどと同様な構造のバッファを指すポインタ
word　FILENO　；ファイル・ハンドル

返り値

D0.L	実際に得られた1行の文字数．(INPPTR＋1).bに入っている値と同じ(0～255)．ファイル終端以降を読み出そうとした場合，－1．
(INPPTR＋0).b	入力最大文字数．あらかじめ呼ぶ側で値を入れておきます．
(INPPTR＋1).b	実際に入力された文字数が入ります（エンドマーク$00は数えません）．
(INPPTR＋2)以降	入力された文字列が入ります．入力最大文字数＋1バイトのエリアを確保しておく必要があります（エンドマーク$00が入るため）．

機能

ファイルからCR＋LFを行末コードとする1行を読み出します．CRのみでは行末コードとして認識されません．

標準入力からの入力の場合，エコーバック，ブレイクチェックを行ないます．

コール

```
move.w  #FILENO,-(SP)
pea     INPPTR
dc.w    $FF1C
addq.l  #6,SP
        :
```

INPPTR:

```
        dc.b    40
        dc.b    0
        ds.b    41
```

参照

$FF0A　gets (INPPTR)
$FF1A　getss (INPPTR)
$FF3D　open (NAMEPTR, MODE)
$FF3F　read (FILENO, DATAPTR, SIZE)

サンプル・プログラム

P. 210　SMPL10

$FF1D fputc(CODE,FILENO)　1.0x/2.0x

引数

word　CODE　(ただし$0000～$00FF)
word　FILENO　；ファイル・ハンドル

返り値

D0.L　異常なしの場合は1(実際に書き込んだ文字数)．書き込めなかった場合は0が返る．

機能

ファイルに1文字書き込みます．
標準出力への出力の場合ブレイクチェックを行ないます．

コール

```
        move.w  #FILENO,-(SP)
        move.w  #CODE,-(SP)
        dc.w    $FF1D
        addq.l  #4,SP
```

参照

$FF02　putchar()
$FF3C　creat (NAMEPTR, ATR)
$FF3D　open (NAMEPTR, MODE)
$FF3F　read (FILENO, DATAPTR, SIZE)

サンプル・プログラム

P. 207　SMPL9

$FF1E fputs(MESPTR,FILENO)　　1.0x/2.0x

引数

long　MESPTR　；文字列を指すポインタ
word　FILENO　；ファイル・ハンドル

返り値

D0.L　実際に書き込んだ文字数．書き込めなかった場合は0が返る．

機能

ファイルにポインタで示した文字列を書き込みます．行末に改行コードなどは付加されません．

ブレイクチェックを行ないます．

コール

```
        move.w  #FILENO,-(SP)
        pea     MESPTR
        dc.w    $FF1E
        addq.l  #6,SP
                :
MESPTR:
        dc.b    'This is fputs test',13,10,0
```

参照

$FF09　print(MESPTR)
$FF3C　creat(NAMEPTR,ATR)
$FF3D　open(NAMEPTR,MODE)
$FF3F　read(FILENO,DATAPTR,SIZE)

サンプル・プログラム

P.210　SMPL10

$FF1F allclose( )　　1.0x/2.0x

引数

なし．

返り値

なし．

機能

現在オープンされているファイルをすべてクローズします．

子プロセスがオープンしたファイルもクローズします．親プロセスがオープンしたファイルはクローズしません．

コール

```
        dc.w    $FF1F
```

参照

$FF3E　close(FILENO)

サンプル・プログラム

P.207　SMPL9

$FF20 super[STACK] 1.0x/2.0x

引数

long　STACK　;システム・スタック・ポインタの値

返り値

D0.L　STACK＝0の場合のみ，前のシステム・スタック・ポインタの値．負の数ならエラーコードです．

機能

★STACKが0の場合

ユーザー・スタック・ポインタの値をシステム・スタック・ポインタにセットして，CPUの特権状態をスーパーバイザ状態に切り替えます．スーパーバイザ状態で動作している間，このファンクションで返されたシステム・スタック・ポインタの値は，再びユーザ状態に戻るときに必要になりますから，ワークエリアなどに保存しておく必要があります．

I/Oポートなど，スーパーバイザ空間をアクセスするためにはこのファンクションでをスーパーバイザ状態に切り替えてからアクセスします．

★STACKが0以外の場合

STACKの値をシステム・スタック・ポインタにセットして，CPUの特権状態をユーザ状態に切り替えます．

コール

```
        clr.l    -(SP)
        dc.w     $FF20
        addq.l   #4,SP
        move.l   D0,SSPBUF
        move.l   USP,A0           *省略可
        move.l   A0,USPBUF        *省略可
                 :
        move.l   USPBUF,A0        *省略可
        move.l   A0,USP           *省略可
        move.l   SSPBUF,-(SP)
        dc.w     $FF20
        addq.l   #4,SP
                 :
USPBUF:                           *省略可
        ds.l     1                *省略可
SSPBUF:
        ds.l     1
```

＊プロセス実行前のSPやSRの内容はHumanによって保存されています．したがって，短いテスト用のプログラムなどならば，スーパーバイザ・モードを解除することなくプロセスを終了しても構わないでしょう．

サンプル・プログラム

P.213　SMPL11

$FF21　fnckey[MD * 256+FNO,BUFFER]　　1.0x/2.0x

引数

word　MD*256+FNO　；MDはこのファンクションの動作モードを指定し，FNOは操作の対象とする定義可能なキーの番号を指定します．

long　BUFFER　；定義可能なキーに設定する文字列を指すポインタ．または，定義可能なキーの内容を読み出すバッファを指すポインタ．

返り値

★ MDが0の場合

●FNOが0の場合

すべての定義可能なキーの内容，20×32＋12×6，計712バイトがBUFFERの指すバッファに返されます．

●FNOが1～20の場合

定義可能なキー1～20（ファンクションキー）の内容，32バイト（31バイト＋$00）がBUFFERの指すバッファに返されます．

●FNOが21～32の場合

定義可能なキー21～32（特殊キー）の内容，6バイト（5バイト＋$00）がBUFFERの指すバッファに返されます．

★ MDが1の場合

なし．

機能

表4

FNOの値	定義可能キー
0	すべての定義可能なキー
1～10	F1～F10
11～20	SHIFT+F1～F10
21	ROLL UP
22	ROLL DOWN
23	INS
24	DEL
25	UP ↑
26	LEFT ←
27	RIGHT →
28	DOWN ↓
29	CLR
30	HELP
31	HOME
32	UNDO

★ MDが0の場合

定義可能キーの内容を読み出します．結果はBUFFERの指すバッファに返されます．

★ MDが1の場合

定義可能なキーの内容を設定します．31行モードのときは最下行にファンクションキーの内容を表示します．

キーの内容としては，それぞれのキーに定義できる文字数以内の文字列を指定します．どのような文字を指定してもかまわないのですが，$FEだけは特殊な扱われ方をします．

$FEに続く7バイトはファンクションキー表示にのみ用いられ，実際にそのキーを押した場合には8文字目以降が入力されます．

```
$FE□□□□□□□□|□□  ～ □  □
 1 2 3 4 5 6 7 8  9 10     31  32文字目
 └─────────────┘ └──────────────────┘
     表示のみ       実際に入力される文字
```

このファンクションコールはCONデバイスによりサポートされます．

コール

```
        pea     BUFFER
        move.w  #MD*256+FNO,-(SP)
        dc.w    $FF21
        addq.l  #6,SP
                :
BUFFER:
        ds.b    712
```

参照

$FF23　conctrl(MD,??)

サンプル・プログラム

P.213　SMPL12

$FF22 knjctrl(MD,??)　　1.0x/2.0x

このファンクションは日本語FPのために用意されています．第3章，『日本語フロント・プロセッサ』を参照してください．

$FF23 conctrl(MD,??)　　1.0x/2.0x

引数

word　MD　；このファンクションの動作モードを指定します．

その他のパラメータは，指定したモードによって意味や数が違ってきます．

返り値

★ MD が14の場合

D0.L　ファンクションキーのモード設定状況

★ MD が16の場合

D0.L　スクリーンモードの設定状況

★ MD がその他の値の場合

なし．

機能

★ MD が 0 の場合

conctrl(0,CODE)

word　CODE

CODEで指定した1バイト・データを表示します．

★ MD が 1 の場合

conctrl(1,MESPTR)

long　MESPTR

MESPTRでポイントした文字列を表示します．

★ MD が 2 の場合

conctrl(2,ATR)

word　ATR

ATRで指定した属性をセットします．

表5

ATRの値	属　性
0	黒
1	水色
2	黄色
3	白
4～7	それぞれの強調
8～11	それぞれのリバース
12～15	それぞれの強調リバース

★ MD が 3 の場合

conctrl(3,X,Y)

word　X,Y

X,Yで指定した座標にカーソルを移動します．

★ MDが4の場合

conctrl(4)

1行下にカーソルを移動します．最下行ではスクロール・アップします．

★ MDが5の場合

conctrl(5)

1行上にカーソルを移動します．先頭行ではスクロール・ダウンします．

★ MDが6の場合

conctrl(6, N)

word　N

N行上にカーソルを移動します．スクロール・ダウンしません．Nに0を指定した場合は1を指定したときと同じ．

★ MDが7の場合

conctrl(7, N)

word　N

N行下にカーソルを移動します．スクロール・アップしません．Nに0を指定した場合は1を指定したときと同じ．

★ MDが8の場合

conctrl(8, N)

word　N

N文字右にカーソルを移動します．Nに0を指定した場合は1を指定したときと同じ．

★ MDが9の場合

conctrl(9, N)

word　N

N文字左にカーソルを移動します．Nに0を指定した場合は1を指定したときと同じ．

★ MDが10の場合

conctrl(10, MOD)

word　MOD

MODにしたがって画面のクリアを行ないます．

表6

MOD	機　　能
0	カーソル位置から最終行左端までクリア
1	ホームからカーソル位置までクリア
2	画面御全体をクリア。カーソルはホームへ

★ MDが11の場合

conctrl(11, MOD)

word　MOD

MODにしたがって現在行のクリアを行ないます．

表7

MOD	機　　能
0	カーソル位置から行の右端までクリア
1	行の左端からカーソル位置までクリア
2	カーソルのある行を一行全部クリア

★ MD が12の場合

```
conctrl(12,N)
word   N
```

カーソル行にN行挿入．Nに0を指定した場合は1を指定したときと同じ．

★ MD が13の場合

```
conctrl(13,N)
word   N
```

カーソル行からN行削除．Nに0を指定した場合は1を指定したときと同じ．

★ MD が14の場合

```
conctrl(14,MD)
word   MD
```

MDにしたがって最下行のファンクションキーの表示状態を設定します．返り値としてD0に前のモードが返ります．スクロール範囲はリセットされます．

表8

MOD	モード
0	ファンクションキー表示。スクロール範囲は0～31になります。
1	シフトファンクションキー表示。スクロール範囲は0～31になります。
2	何も表示しません。スクロール範囲は0～31になります。
3	普通の行とします。スクロール範囲は0～32になります。
－1	何もしません。前のモードがD0に返るだけです。

★ MD が15の場合

```
conctrl(15,YS,YL)
word   YS,YL
```

スクロール・エリアスタート行YS，スクロール行数YLを指定して，スクロール範囲を設定します．YS＋YLの値はファンクションキーを表示している状態では31まで，表示していない状態では32までです．

実行後は絶対座標（0,YS）が新しいホーム位置となり，その後は論理座標の（0,0）となります．カーソルは新しいホーム位置に移動します．

★ MD が16の場合

```
conctrl(16,MOD)
word   MOD
```

MODにしたがってスクリーンモードを変更します．返り値としてD0に前のモードが返ります．

表9

MOD	スクリーンモード
0	高解像度モード768＊512グラフィックなし
1	高解像度モード768＊512グラフィック16色
2	高解像度モード512＊512グラフィックなし
3	高解像度モード512＊512グラフィック16色
4	高解像度モード512＊512グラフィック256色
5	高解像度モード512＊512グラフィック65536色
－1	何もしません。前のモードがD0に返るだけです。

★ MDが17の場合

conctrl(17)

カーソルを表示するモードにします.

★ MDが18の場合

conctrl(18)

カーソルを表示しないモードにします.

このファンクションコールはCONデバイスによりサポートされます.

コール

(例はMDが16の場合)

```
        clr.w     -(SP)
        move.w    #16,-(SP)
        dc.w      $FF23
        addq.l    #4,SP
```

サンプル・プログラム

P.213　SMPL12

$FF24 keyctrl(MD,??)　　1.0x/2.0x

引数

word　MD　；このファンクションの動作モードを指定します．
その他のパラメータは，指定したモードによって意味や数が違ってきます．

返り値

★ MDが0の場合
D0.L　キーコード（ただし$00000000～$000000FF）

★ MDが1の場合
D0.L　キーコード（ただし$00000000～$000000FF）

★ MDが2の場合
D0.L　シフトキーの状態

表10

b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
全角	ヒラガナ	INS	CAPS	コード	ローマ	カナ	opt2	opt1	ctrl	shift

ビットが1のキーが押されている/ロックされている（その他のビットは意味を持たない）．

★ MDが3の場合
D0.L　指定したキーコードグループのキー状態
IOCS　$04 B_BITSNS参照

機能

★ MDが0の場合
keyctrl(0)
キー入力を行ない，キーコードをD0に返します．IOCS　$00 B_KEYINPに相当．

★ MDが1の場合
keyctrl(1)
キーコードの先読みを行ない，結果をD0に返します．IOCS　$01 B_KEYSNSに相当．

★ MDが2の場合
keyctrl(2)
シフトキー状態をセンスし，結果をD0に返します．IOCS　$02 B_SFTSNSに相当．

★ MDが3の場合
keyctrl(3,GROUP)
word　GROUP
指定キーグループのセンスを行ない，状態をD0に返します．IOCS　$04 B_BITSNSに相当．

★ MDが4の場合
keyctrl(4,INSMODE)
word　INSMODE
INS キーのon（$00FF）/off（$0000）を設定します．返り値はありません．
このファンクションコールはCONデバイスによりサポートされます．

コール

（例は[illegible]が3の場合）

```
        move.w   #$E,-(SP)
        move.w   #3,-(SP)
        dc.w     $FF24
        addq.l   #4,SP
```

サンプル・プログラム

P.216　SMPL13

$FF25　intvcs(INTNO,JOBADR)　　1.0x/2.0x

引数

word　INTNO　；各種ベクターを示す値
long　JOBADR　；割り込み処理アドレス

返り値

D0.L　設定前のベクタ

機能

各種ベクタ（ハード/ソフト割り込み，IOCS，ファンクションコール）を設定します．

★ INTNO が$00～$FF の場合

$00000000～$000003FFにあるハード/ソフト割り込みベクターを設定します．INTNOはベクタ番号で，実際にベクタが格納されるのはINTNO×4のアドレスです．

★ INTNO が$100～$1FF の場合

IOCSコール，$00～$FFまでの処理アドレスを設定します．

★ INTNO が$FF00～$FFFF の場合

ファンクションコール，$FF00～$FFFFの処理アドレスを設定します．

コール

```
        pea      JOBADR
        move.w   #INTNO,-(SP)
        dc.w     $FF25
        addq.l   #6,SP
```

参照

$FF35　intvcg(INTNO)

サンプル・プログラム

P.216　SMPL14

$FF26 pspset(PSPADR)　　1.0x/2.0x

引数

long　PSPADR　;PSPを作成するアドレス

返り値

なし

機能

PSPADRで示すアドレスからPSP（正確にはプロセス管理テーブル）を作成し，制御を移します．PSPADRには$FF48 mallocの返り値（メモリ管理ポインタの先頭＋$10）を指定します．

このファンクションコールは，ひとつのプロセスから別のプロセスをメモリ上に作成する場合に使います．

作成されるPSPは次のようなものです．

●環境，CTRL ＋ C でのリターン・アドレス，エラーによるリターン・アドレスは現在のものを引き継ぐ．
●プロセス終了時のリターン・アドレスは$FF26 pspsetをコールした次のアドレス．
●コマンドラインのアドレスは$00000000．
●ファイル・ハンドルの使用状況は，すべて0．
●bss／ヒープ／スタックはすべて0．したがって，プロセスとしての中身はまったくない．
●新しいプロセスは「親あり」に設定．
●「execされたファイル」関係は空白．

その他のパラメータはきちんと設定されます．

また，現在のプロセスのPSPには，このコールで作成したPSPの先頭アドレスが子プロセスのPSPとして登録されます．

コール

```
move.l  #PSPADR,-(SP)
dc.w    $FF26
addq.l  #4,SP
```

参照

$FF48　malloc(LEN)
$FF50　setpdb(PDBADR)
$FF51　getpdb()

サンプル・プログラム

P.219　SMPL15

$FF27 gettim2[ ]　　1.0x/2.0x

引数

なし

返り値

D0.l　現在の時刻

```
b31←――――――――――――→ b0
00000000 000hhhhh 00mmmmmm 00ssssss
        bit16～20(hhhhh)    0～23時
        bit 8～13(mmmmmm)   0～59分
        bit 0～ 5(ssssss)   0～59秒
```

機能

現在の時刻をD0に返します.

IOCSの$52 TIMEBCD, $57 TIMEBIN, $59 TIMECNV, $5B TIMEASCで使用されているバイナリ形式の時刻フォーマットと同一なデータ形式であり, $FF2C gettimeのデータ形式とは異なります.

コール

```
dc.w    $FF27
```

参照

$FF28　settim2(TIME)
$FF2A　getdate()
$FF2B　setdate(DATE)
$FF2C　gettime()
$FF2D　settime(TIME)

サンプル・プログラム

P.202　SMPL5

$FF28 settim2(TIME)　　1.0x/2.0x

引数

long　TIME　；設定する時刻

b31←――――――――――→ b0

00000000 0000hhhh 00mmmmmm 00ssssss

bit16～20(hhhhh)　0～23時
bit 8～13(mmmmmm)　0～59分
bit 0～ 5(ssssss)　0～59秒

返り値

D0.L　負の数の場合はエラーコード

機能

時刻をTIMEの値にしたがって設定します．

IOCSの$52TIMEBCD，$57TIMEBIN，$59TIMECNV，$5BTIMEASCで使用されているバイナリ形式の時刻フォーマットと同一なデータ形式であり，$FF2D settimeのデータ形式とは異なります．

コール

```
move.l  #TIME,-(SP)
dc.w    $FF28
addq.l  #4,SP
```

参照

$FF27　gettim2(TIME)
$FF2A　getdate()
$FF2B　setdate(DATE)
$FF2C　gettime()
$FF2D　settime(TIME)

サンプル・プログラム

P.205　SMPL5

$FF29 namests(FILE,BUFFER) 1.0x/2.0x

引数

long FILE ；ファイル・ネームを指すポインタ
long BUFFER ；結果が入るバッファを指すポインタ

返り値

D0.L 負の数の場合はエラーコード

(BUFFER＋0).b 0ならワイルドカード指定なし，$FFならファイル指定なし．それ以外はワイルドカード指定あり．

(BUFFER＋1).b ドライブ番号．A＝0,B＝1…

(BUFFER＋2).b
: パス文字列．エンドマークは$00．
(BUFFER＋66).b (65バイト)

(BUFFER＋67).b
: ファイル・ネーム．8文字．8文字以下の場合は$20（スペース）で埋められる．
(BUFFER＋74).b

(BUFFER＋75).b
: 拡張子．3文字．3文字以下の場合は$20（スペース）で埋められる．
(BUFFER＋77).b

(BUFFER＋78).b
: ファイル名残り．10文字．10文字以下の場合は$00で埋められる．
(BUFFER＋87).b

機能

ファイル・ネームをバッファに展開します．バッファは88バイト必要です．

動作は$FF37 nameckと似ていますが，ドライブネームを番号に直して返したり，ファイル名の先頭8文字と残りの10文字を分離して返すなど，こちらの方がよりハードウェア寄りの情報が得られます．

コール

```
        pea     BUFFER
        pea     FILE
        dc.w    $FF29
        addq.l  #8,SP
                :
FILE:
        dc.b    'A:¥COMMAND.X',0
BUFFER:
        ds.b    88
```

参照

$FF37 nameck(FILE,BUFFER)

サンプル・プログラム

P.220 SMPL16

$FF2A getdate( )　　1.0x/2.0x

引数

なし.

返り値

D0.L　現在の日付.

```
b31←―――――――――――――――――→ b0
00000000 00000www yyyyyyym mmmddddd

bit16～18(www)      0～6 (日, 月…土曜日)
bit 9～15(yyyyyyy)  0～119(+1980)年
bit 5～ 8(mmmm)     1～12月
bit 0～ 4(ddddd)    1～31日
```

機能

現在の日付をD0に返します.

コール

```
        dc.w    $FF2A
```

参照

$FF27　gettim2()
$FF28　settim2(TIME)
$FF2B　setdate(DATE)
$FF2C　gettime()
$FF2D　settime(TIME)

サンプル・プログラム

P.222　SMPL17

$FF2B setdate(DATE)　　1.0x/2.0x

引数

word　DATE　;設定する日付.

```
b15←―――――→ b0
yyyyyyym mmmddddd

bit 9～15(yyyyyyy)  0～119(+1980)年
bit 5～ 8(mmmm)     1～12月
bit 0～ 4(ddddd)    1～31日
```

返り値

D0.L　負の数の場合はエラーコード

機能

日付をDATEの値にしたがって設定します.

コール

```
        move.w  #DATE,-(SP)
        dc.w    $FF2B
        addq.l  #2,SP
```

参照

$FF27 gettim2()

$FF28 settim2(TIME)

$FF2A getdate()

$FF2C gettime()

$FF2D settime(TIME)

サンプル・プログラム

P. 222 SMPL17

$FF2C gettime[ ] 1.0x/2.0x

引数

なし

返り値

D0.w 現在の時刻

b15←――――→ b0

hhhhh mmm mmm sssss

bit11～15(hhhhh) 0～23時

bit 5～11(mmmmmm) 0～59分

bit 0～ 4(ssssss) (0～29)×2秒

機能

現在の時刻をD0に返します．

$FF27 gettim2とはデータ形式が異なります．秒の位が2秒単位で戻ってくることに注意してください．実際の秒を得るためには，返り値の秒の位を2倍する必要があります．

コール

```
dc.w $FF2C
```

参照

$FF27 gettim2()

$FF28 settim2(TIME)

$FF2A getdate()

$FF2B setdate(DATE)

$FF2D settime(TIME)

サンプル・プログラム

P. 222 SMPL17

$FF2D settime(TIME) 1.0x/2.0x

引数

```
word      TIME      ；設定する時刻
        b15←──────→ b0
        hhhhh mmm mmm sssss

            bit11～15(hhhhh)     0～23時
            bit 5～11(mmmmmm)    0～59分
            bit 0～ 4(ssssss)    (0～29)×2秒
```

返り値

D0.L　　負の数の場合はエラーコード

機能

時刻をTIMEの値にしたがって設定します．

$FF28 settim2とはデータ形式が異なります．秒の位を2秒単位で設定する必要があることに注意してください．実際の秒を設定するためには，2で割った値を設定する必要があります．

コール

```
        move.w  #TIME,-(SP)
        dc.w    $FF2D
        addq.l  #2,SP
```

参照

$FF27　gettim2()
$FF28　settim2(TIME)
$FF2A　getdate()
$FF2B　setdate(DATE)
$FF2C　gettime()

サンプル・プログラム

P.222　SMPL17

$FF2E verify(FLG) 1.0x/2.0x

引数

```
word      FLG      ;ベリファイフラグ
                   =0   ベリファイしない
                   =1   ベリファイする
```

返り値

なし．

機能

ベリファイフラグを設定します．ベリファイフラグを1に設定すると，ディスクの入出力時にデータのベリファイを行ないます．

コール

```
        move.w  #FLG,-(SP)
        dc.w    $FF2E
        addq.l  #2,SP
```

参照

$FF54 verifyg()

サンプル・プログラム

P.246 SMPL33

$FF2F dup0(FILENO,NEWNO) 1.0x/2.0x

引数

word FILENO ;ファイル・ハンドル

word NEWNO ;FILENOを強制複写するファイル・ハンドル(0～4).

返り値

D0.L NEWNOハンドルのコピー前の値. 負の数の場合はエラーコード.

機能

FILENOのファイル・ハンドルをNEWNOのファイル・ハンドルに強制複写します. NEWNOのとり得る値が0～4である点に注意してください. それ以上を指定する場合は$FF46 dup2を使います.

Humanの用意する0～4の標準ファイル・ハンドルのデフォルト値はシステム・ワークに保存されています. FILENOに0～4を指定した場合, NEWNOにコピーされるのはデフォルト値です. したがって, dup0(5,0)のような操作を行なって標準入力ハンドルへハンドル5を複写した後, dup0(0,0)を実行すれば標準入力ハンドルをデフォルト・デバイスに戻すことができます.

このファンクションコールはCTTYコマンド/リダイレクトを実現するのに使われています.

コール

```
        move.w  #NEWNO,-(SP)
        move.w  #FILENO,-(SP)
        dc.w    $FF2F
        addq.l  #4,SP
```

参照

$FF45 dup(FILENO)

$FF46 dup2(FILENO,NEWNO)

サンプル・プログラム

P.239 SMPL29

$FF30 vernum( ) 1.0x/2.0x

引数

なし.

返り値

D0.L　上位16ビット = '68' ($3638)
　　　下位16ビット = 整数部*256+小数部

機能

Human68Kのバージョン番号を返します.

コール

```
        dc.w    $FF30
```

サンプル・プログラム

P.224 SMPL18

$FF31 keeppr(PRGLEN,CODE) 1.0x/2.0x

引数

long　PRGLEN　；常駐するプロセスの大きさ（バイト）
word　CODE　　；終了コード

返り値

なし（戻ってこない）.

機能

プログラムの先頭からPRGLENで示される大きさを残して，プロセスを常駐終了します.

CODEは終了コードで，通常は0ですが，エラーが発生したことなどを親プロセスに知らせる場合などには適当な値を設定しておきます.

コール

```
        move.w  #CODE,-(SP)
        move.l  #PRGLEN,-(SP)
        dc.w    $FF31
```

参照

$FF00　exit()
$FF4C　exit2(CODE)
$FF4D　wait()

サンプル・プログラム

P.216 SMPL14

$FF32 getdpb[DRIVE,DPBPTR] 1.0x/2.0x

引数

word　DRIVE　;ドライブ番号．カレント・ドライブ＝0, A＝1, B＝2…

long　DPBPTR　;DPB（ドライブパラメータ・ブロック）をコピーするバッファを指すポインタ．

返り値

D0.L　負の数の場合はエラーコード

(DPBPTR＋0).b　装置番号．A=0, B=1, …

(DPBPTR＋1).b　装置ドライバで使うユニット番号

(DPBPTR＋2).w　1セクタ当たりのバイト数

(DPBPTR＋4).b　1クラスタあたりのセクタ数−1

(DPBPTR＋5).b　クラスタ←→セクタのシフトカウント

(DPBPTR＋6).w　FATの先頭セクタ番号

(DPBPTR＋8).b　FAT領域の個数

(DPBPTR＋9).b　FATの占めるセクタ数（コピーの分は別）

(DPBPTR＋10).w　ルートディレクトリの個数

(DPBPTR＋12).w　データ部の先頭セクタ番号

(DPBPTR＋14).w　総クラスタ番号＋1

(DPBPTR＋16).w　ルートディレクトリの先頭セクタ番号

(DPBPTR＋18).l　装置ドライバへのポインタ

(DPBPTR＋22).b　メディア・バイト

(DPBPTR＋23).b　DPB 使用フラグ（−1でアクセスなし）

(DPBPTR＋24).l　次のDPBへのポインタ

(DPBPTR＋28).w　カレント・ディレクトリのクラスタ番号（0はルートを表わす）

(DPBPTR＋30).b
　:　カレント・ディレクトリの文字バッファ（64バイト）
(DPBPTR＋93).b

機能

指定したドライブ番号のDPB（ドライブパラメータブロック）をバッファにコピーします．

バッファは94バイト必要です．

Human1.0xでは/で区切られたパス・リストが返ってきますが，Human2.0xでは￥で区切られたパス・リストが返ってきます．

コール

```
        pea     DPBPTR
        move.w  #DRIVE,-(SP)
        dc.w    $FF32
        addq.l  #6,SP
                :
DPBPTR:
        ds.b    94
```

参照

$FF36　dskfre (DRIVE, BUFFER)

サンプル・プログラム

P. 225　SMPL19

$FF33　breakck(FLG)　　1.0x

引数

word　　FLG　　;ブレークフラグ

　　　　　　＝0　　指定のファンクションコールのみチェックする

　　　　　　＝1　　すべてのファンクションコールでチェックする

　　　　　　＝－1　設定状況を見るだけ

返り値

D0.L　　ブレークフラグの設定状況

　　　　　　＝0　指定のファンクションコールでのみブレークチェックする

　　　　　　＝1　すべてのファンクションコールでブレークチェックする

機能

ブレークフラグを設定/参照します.

コール

```
        move.w  #FLG,-(SP)
        dc.w    $FF33
        addq.l  #2,SP
```

サンプル・プログラム

P. 226　SMPL20

$FF34 drvxchg(OLD,NEW) 1.0x/2.0x

引数

word OLD ;カレント・ドライブ=0, A=1, B=2

word NEW ; 〃

返り値

D0.L 負の数の場合はエラーコード

機能

OLDのドライブとNEWのドライブを入れ替えます.

OLDとしてAドライブを, NEWとしてBドライブを指定してこのファンクションを実行した場合, Bドライブであったドライブが A として, Aドライブであったドライブが B として扱われるようになります.

コール

```
move.w  #NEW,-(SP)
move.w  #OLD,-(SP)
dc.w    $FF34
addq.l  #4,SP
```

参照

$FF5F assign(MD,???) 2.0x

サンプル・プログラム

P.226 SMPL21

$FF35 intvcg(INTNO) 1.0x/2.0x

引数

word INTNO ;割り込みベクタを示す値.

返り値

D0.L ベクタの値

機能

★ INTNO が$0000～$00FF の場合

$00000000～$000003FFにあるハード/ソフト割り込みベクターを読み出します. INTNOはベクタ番号で, 実際にベクタが格納されているのはINTNO×4のアドレスです.

★ INTNO が$FF00～$FFFFの場合

ファンクションコール, $FF00～$FFFFの処理アドレスを読み出します.

コール

```
move.w  #INTNO,-(SP)
dc.w    $FF35
addq.l  #2,SP
```

参照

$FF25 intvcs(INTNO,JOBADR)

サンプル・プログラム

P.216 SMPL14

$FF36 dskfre(DRIVE,BUFFER) 1.0x/2.0x

引数

word　DRIVE　；カレント・ドライブ＝0, A＝1, B＝2, …
long　BUFFER　；結果が入るバッファを指すポインタ

返り値

D0.L　使用可能なバイト数．負の数の場合はエラーコード．
(BUFFER＋0).w　使用可能なクラスタ数
(BUFFER＋2).w　総クラスタ数
(BUFFER＋4).w　1クラスタあたりのセクタ数
(BUFFER＋6).w　1セクタあたりのバイト数

機能

ディスクの残り容量を返します．
バッファは8バイト必要です．

コール

```
        pea     BUFFER
        move.w  #DRIVE, -(SP)
        dc.w    $FF36
        addq.l  #6, SP
                :
BUFFER:
        ds.w    4
```

参照

$FF32　getdpb(DRIVE, DPBPTR)

サンプル・プログラム

P.227　SMPL22

$FF37 nameck(FILE,BUFFER) 1.0x/2.0x

引数

long　FILE　；ファイル・ネームを指すポインタ
long　BUFFER　；結果が入るバッファを指すポインタ

返り値

D0.L　$00ならワイルドカード指定なし．
　　$FFならファイル指定なし．
　　それ以外の正の数の場合はワイルドカード指定あり．
　　負の数の場合はエラーコード

```
(BUFFER+0).b
      :          ドライブネーム．"A:" など．
(BUFFER+1).b     (2バイト)
(BUFFER+2).b
      :          パスネーム．エンドマークは$00．
(BUFFER+66).b    (64+1バイト)
(BUFFER+67).b
      :          ファイル・ネーム．エンドマークは$00．
(BUFFER+85).b    (18+1バイト)
(BUFFER+86).b
      :          拡張子．ピリオド含む．エンドマークは
(BUFFER+91).b    $00 (1+3+1バイト)．
```

機能

ファイル・ネームをバッファに展開します．バッファは92バイト必要です．
エラーリターンした場合はBUFFERの内容は意味を持ちません．
Human1.0xでは/で区切られたパス・リストが返ってきますが，Human2.0xでは¥で区切られたパス・リストが返ってきます．
$FF29 namestsと動作が似ていますが，こちらの方がよりOS寄りの情報を得られます．

コール

```
        pea     BUFFER
        pea     FILE
        dc.w    $FF37
        addq.l  #8,SP
                :
FILE:
        dc.b    'A:¥COMMAND.X',0
BUFFER:
        ds.b    92
```

参照

$FF29 namests(FILE,BUFFER)

サンプル・プログラム

P.233 SMPL26

$FF39 mkdir(NAMEPTR) 1.0x/2.0x

引数

long　　NAMEPTR ;ディレクトリの名前を指すポインタ

返り値

D0.L　　負の数の場合はエラーコード

機能

ディレクトリを作成します.

コール

```
        pea     NAMEPTR
        dc.w    $FF39
        addq.l  #4,SP
                :
NAMEPTR:
        dc.b    'A:¥BIN',0
```

参照

$FF3A　rmdir(NAMEPTR)
$FF3B　chdir(NAMEPTR)

サンプル・プログラム

P.229　SMPL23

$FF3A rmdir(NAMEPTR) 1.0x/2.0x

引数

long　　NAMEPTR ;ディレクトリの名前を指すポインタ

返り値

D0.L　　負の数の場合はエラーコード

機能

ディレクトリを削除します. カレント・ディレクトリや, 中にファイルが1つでも存在するディレクトリは削除できません.

コール

```
        pea     NAMEPTR
        dc.w    $FF3A
        addq.l  #4,SP
                :
NAMEPTR:
        dc.b    'A:¥BIN',0
```

参照

$FF39　mkdir(NAMEPTR)
$FF3B　chdir(NAMEPTR)

サンプル・プログラム

P.230　SMPL24

$FF3B chdir(NAMEPTR)　1.0x/2.0x

引数

long　NAMEPTR ;ディレクトリの名前を指すポインタ

返り値

D0.L　負の数の場合はエラーコード

機能

カレント・ディレクトリを変更します.

コール

```
        pea     NAMEPTR
        dc.w    $FF3B
        addq.l  #4,SP
                :
NAMEPTR:
        dc.b    'A:¥BIN',0
```

参照

$FF39　mkdir(NAMEPTR)

$FF3A　rmdir(NAMEPTR)

サンプル・プログラム

P.229　SMPL23

Human68Kの2HDディスク

Human68Kの2HDディスクはPC-9801のMS-DOS 2HDディスクと互換性があります. よって, ASCIIファイルは当然として, バイナリ・ファイルも双方で読み書きできます. ただし, Human68Kではファイル名に小文字が許されていますが, MS-DOSの方ではディレクトリを取ると表示されるだけで, アクセスできません. CASE.Xなどで大文字にするとアクセスできるようになります.

DISKCOPYコマンドはそれぞれ同じ働きをします. よって, Human68KのディスクをDISKCOPY.EXEでコピーできますし, MS-DOSのディスクをDISKCOPY.Xでコピーすることもできます.

ただし, FORMATコマンドはIPLが異なるため, 基本的には互換性がありません. FORMAT.EXEでフォーマットしたディスクにHUMAN.SYSをコピーした場合は起動しませんし, 逆もうまくいきません. ただし, うまくいかないのは違うシステムでフォーマットしたディスクに, システムファイルをコピーして立ち上げディスクとして使う場合です. ですから, データ・ディスクなどは違うシステムでフォーマットしても動作します.

$FF3C create[NAMEPTR,ATR]　　1.0x/2.0x

引数

```
long    NAMEPTR ; ファイル・ネームを指すポインタ
word    ATR     ; ファイルの属性
```

返り値

D0.L　ファイル・ハンドル．負の数の場合はエラーコード．

機能

ファイルを作成します．同じ名前のファイルが存在する場合も新しく作り直します．

ATRの各ビットを立てた場合，以下のようなファイルの属性が指定されます．複数のビットを立てることも可能です．

bit5	通常のファイナル
bit4	ディレクトリ
bit3	ボリュームID
bit2	システム・ファイル
bit1	隠しファイル
bit0	読み込み専用ファイル

(その他のビットは意味を持たない)

返り値としてD0に返されるファイル・ハンドルは以降のファイル・アクセスに必要ですから，ワークエリアなどに保存しておくのが普通です．

コール

```
        move.w  #ATR,-(SP)
        pea     NAMEPTR
        dc.w    $FF3C
        addq.l  #6,SP
                :
NAMEPTR:
        dc.b    'A:¥DOC¥READ_ME.DOC',0
```

参照

$FF3D	open(NAMEPTR,MODE)	
$FF3E	close(FILENO)	
$FF5A	creattmp(NAMEPTR,MODE)	2.0x
$FF5B	create2(NAMEPTR,MODE)	2.0x

サンプル・プログラム

P.207　SMPL9

$FF3D open(NAMEPTR,MODE) 1.0x

引数

long　NAMEPTR ；ファイル・ネームを指すポインタ
word　MODE ；ファイルをオープンするモード

返り値

D0.L　ファイル・ハンドル．負の数の場合はエラーコード．

機能

★ MODE が 0 の場合
読み込みオープン

★ MODE が 1 の場合
書き込みオープン

★ MODE が 2 の場合
読み込み/書き込みオープン

★ MODE が$100 の場合
読み込みオープン（辞書用の特殊ハンドラを返す．ユーザーは使用禁止）

★ MODE が$101 の場合
書き込みオープン（辞書用の特殊ハンドラを返す．ユーザーは使用禁止）

★ MODE が$102 の場合
読み込み/書き込みオープン（辞書用の特殊ハンドラを返す．ユーザーは使用禁止）

このファンクションは既に存在するファイルに対して処理を行なうものであり，ファイルを新しく作成できないことに注意してください．新しいファイルを作成する場合は$FF3C createを使います．

返り値としてD0に返されるファイル・ハンドルは以降のファイル・アクセスに必要ですから，ワークエリアなどに保存しておくのが普通です．

コール

```
        move.w  #MODE, -(SP)
        pea     NAMEPTR
        dc.w    $FF3D
        addq.l  #6, SP
                :
NAMEPTR:
        dc.b    'A:¥DOC¥READ_ME.DOC', 0
```

参照

$FF3C　create(NAMEPTR, MODE)
$FF3E　close(FILENO)
$FF5A　creattmp(NAMEPTR, MODE)　2.0x
$FF5B　create2(NAMEPTR, MODE)　2.0x

サンプル・プログラム

P.207　SMPL9

$FF3E close(FILENO) 1.0x/2.0x

引数

word FILENO ；ファイル・ハンドル

返り値

D0.L 負の数の場合はエラーコード．

機能

ファイルをクローズします．オープン/クリエイトしたファイルは必ずクローズするようにしてください．オープン/クリエイトしたファイルすべてをいちどにクローズしたい場合は$FE1F allcloseが便利です．

コール

```
move.w  #FILENO,-(SP)
dc.w    $FF3E
addq.l  #2,SP
```

参照

$FF1F allclose()
$FF3C create(NAMEPTR,MODE)
$FF3D open(NAMEPTR,MODE)
$FF5A creattmp(NAMEPTR,MODE) 2.0x
$FF5B create2(NAMEPTR,MODE) 2.0x

サンプル・プログラム

P.231 SMPL25

$FF3F read(FILENO,DATAPTR,SIZE) 1.0x/2.0x

引数

word FILENO ；ファイル・ハンドル
long DATAPTR ；データを読み込むバッファを指すポインタ
long SIZE ；読み込むバイト数

返り値

D0.L 実際に読み込んだバイト数．負の数の場合はエラーコード．

機能

FILENOで指定するファイル・ハンドルからSIZEで指定するバイト数だけDATAPTRの指すバッファに読み込みます．

ファイルの終端まで読み出したことは，返り値のD0の値が指定した読み込みバイト数SIZEより小さかったことで判断できます．

バッファはSIZEバイト以上必要です．

コール

```
move.l  #SIZE,-(SP)
pea     DATAPTR
move.w  #FILENO,-(SP)
dc.w    $FF3F
```

```
        lea     10(SP),SP
                :
DATAPTR:
        ds.b    SIZE
```

参照

$FF1A fgetss(INPPTR)
$FF1B fgetc(FILENO)
$FF1C fgets(BUFFER,FILENO)
$FF3D open(NAMEPTR,MODE)

サンプル・プログラム

P.231 SMPL25

$FF40 write(FILENO,DATAPTR,SIZE) 1.0x/2.0x

引数

word FILENO ；ファイル・ハンドル
long DATAPTR ；ファイルに書き込むバッファを指すポインタ
long SIZE ；書き込むバイト数

返り値

D0.L 実際に書き込んだバイト数．負の数の場合はエラーコード．

機能

DATAPTRの指すバッファのデータを，FILENOで指定するファイル・ハンドルへ，SIZEで指定するバイト数だけ書き込みます．

コール

```
        move.l  #SIZE,-(SP)
        pea     DATAPTR
        move.w  #FILENO,-(SP)
        dc.w    $FF40
        lea     10(SP),SP
                :
DATAPTR:
        dc.b    xx,xx,xx,…
```

参照

$FF1D fputc(CODE,FILENO)
$FF1E fputs(BUFFER,FILENO)
$FF3C create(NAMEPTR,MODE)
$FF3D open(NAMEPTR,MODE)
$FF5A creattmp(NAMEPTR,MODE) 2.0x
$FF5B create2(NAMEPTR,MODE) 2.0x

サンプル・プログラム

P.231 SMPL25

$FF41 delete(NAMEPTR) 1.0x/2.0x

引数

long　NAMEPTR ;ファイル・ネームへのポインタ

返り値

D0.L　負の数の場合はエラーコード

機能

ファイルを削除します．

削除に失敗するのはおもに次のような場合です．

- ワイルドカードでファイルを指定した場合．
- 読み出し専用属性が付いたファイルを消去しようとした場合．
- ディレクトリを消去しようとした場合．

ディレクトリを消去する場合は$FF3A rmdirを使います．

コール

```
        pea     NAMEPTR
        dc.w    $FF41
        addq.l  #4,SP
                :
NAMEPTR:
        dc.b    'A:READ_ME.BAK',0
```

参照

$FF3A　rmdir(NAMEPTR)

サンプル・プログラム

P.233　SMPL26

ファイル・アトリビュート$00?

ファンクション・コールの$FF3C creat, $FF3D open, $FF5A creattmpなどでは，オープン/クリエイトするファイルのアトリビュートを指定しますが，このときすべてのビットが0である$00をアトリビュートとして指定したらどうなるでしょうか？

答えは，「$20(通常のファイル)を指定したのと同じ」でした．Human内部でアトリビュートの値をチェックして，$00のときは$20に置き換える処理を行なっているからです．

なお，この件について知っているからといって，メリットはまったくありません．試してみるのは構いませんが，実際にプログラミングに応用したりはしないでください．

$FF42　seek(FILENO,OFFSET,MODE)　1.0x/2.0x

引数

word	FILENO	；ファイル・ハンドル
long	OFFSET	；ポインタを移動するオフセット
word	MODE	；シークモード

返り値

D0.L　現在の先頭からのオフセット．負の数の場合はエラーコード．

機能

ファイル中の，次に読み出し/書き込みを行なうポイントを移動します．

★ MODE が 0 の場合

ファイルの先頭からオフセットの位置に移動する．

★ MODE が 1 の場合

現在の位置からオフセットを加えた位置まで移動する．

★MODEが 2 の場合

ファイルの終わりにオフセットを加えた位置に移動する．

このファンクションを使うことによって，ファイルへのAPPEND，ランダム・ファイル操作などが実現できます．

シークした結果のポインタの位置が返ってくることを利用して，MODEに 2，オフセットに 0 を指定してこのコールを呼び出し，ファイルのサイズを得るというテクニックもしばしば使われます．

コール

```
move.w  #MODE,-(SP)
move.l  #OFFSET,-(SP)
move.w  #FILENO,-(SP)
dc.w    $FF42
addq.l  #8,SP
```

参照

$FF1A　fgetss(INPPTR)
$FF1B　fgetc(FILENO)
$FF1C　fgets(BUFFER,FILENO)
$FF1D　fputc(CODE,FILENO)
$FF1E　fputs(BUFFER,FILENO)
$FF3C　create(NAMEPTR,MODE)
$FF3D　open(NAMEPTR,MODE)
$FF3F　read(FILENO,DATAPTR,SIZE)
$FF40　write(FILENO,DATAPTR,SIZE)

サンプル・プログラム

P.231　SMPL25

$FF43 chmod(NAMEPTR,ATR) 1.0x/2.0x

引数

```
long    NAMEPTR ; ファイル・ネームを指すポインタ
word    ATR     ; ファイルの属性
```

返り値

D0.L 指定ファイルの属性. 負の数の場合はエラーコード

機能

指定したファイルの属性を変更します.

ATRの各ビットを立てた場合, 以下のようなファイルの属性が指定されます. 複数のビットを立てることも可能です.

bit5	通常のファイル
bit4	ディレクトリ
bit3	ボリュームID
bit2	システム・ファイル
bit1	隠しファイル
bit0	読み込み専用ファイル

(その他のビットは意味を持たない)

ATRに－1を指定すると属性の読み出しのみ行ない, 変更は行ないません.

コール

```
        move.w  #ATR,-(SP)
        pea     NAMEPTR
        dc.w    $FF43
        addq.l  #6,SP
                :
NAMEPTR:
        dc.b    'A:¥DOC¥READ_ME.DOC',0
```

参照

$FF3C create(NAMEPTR,MODE)

サンプル・プログラム

P.237 SMPL27

$FF44 ioctrl(MD,??) 1.0x/2.0x

引数

word　　MD　　；このファンクションの動作モードを指定します.

その他のパラメータは，指定したモードによって意味や数が違ってきます.

返り値

★ MD が 0 の場合

D0.L　　装置情報

★ MD が 1 の場合

D0.L　　前の装置情報

★ MD が2, 3, 4, 5の場合

D0.L　　読み込んだ/書き込んだバイト数

★ MD が6, 7の場合

D0.L　　ステータス.（$FF＝可，$00＝不可）

機能

デバイス・ドライバのダイレクト入出力をします. MDが0, 1, 2, 3, 6, 7の場合に指定するFILENOでオープンされているデバイスは，そのドライバにおいてIOCTRLフラグがonになっているものに限られます. Human68kに標準で添付されているデバイス・ドライバの中では，"PCMDRV.SYS"がIOCTRL可能なデバイスとなっています.

★ MD が 0 の場合

ioctrl(0, FILENO)

word　FILENO

FILENOのハンドルの装置情報を返します.

★ MD が 1 の場合

ioctrl(1, FILENO, DT)

word　FILENO

word　DT

FILENOのハンドルに装置情報DTをセットします.

★ MD が 2 の場合

ioctrl(2, FILENO, PTR, LEN)

word　FILENO

long　PTR

long　LEN

FILENOのファイル・ハンドルからLENバイト，PRTの指すバッファに読み込みます.

★ MD が 3 の場合

ioctrl(3, FILENO, PTR, LEN)

word　FILENO

long　PTR

long　LEN

PRTの指すバッファからLENバイト，FILENOのファイル・ハンドルへ書き込みます.

★ MD が 4 の場合

ioctrl(4, DRIVE, PTR, LEN)

word　DRIVE　；カレント・ドライブ＝0, A＝1, B＝2, …

```
long  PTR
long  LEN
```
DRIVEで示されるドライブに対して，MD=2と同じ動作を行ないます．

★ MDが5の場合
```
ioctrl(5,DRIVE,PTR,LEN)
word   DRIVE   ;カレント・ドライブ=0,A=1,B=2,…
long   PTR
long   LEN
```
DRIVEで示されるドライブに対して，MD=3と同じ動作を行ないます．

★ MDが6の場合
```
ioctrl(6,FILENO)
word   FILENO
```
ハンドルの入力ステータスを返します．

★ MDが7の場合
```
ioctrl(7,FILENO)
word   FILENO
```
ハンドルの出力ステータスを返します．

コール

(例はMDが3の場合)
```
        move.l   #LEN,-(SP)
        pea      PTR
        move.w   #3,-(SP)
        dc.w     $FF44
        addq.l   #8,SP
                 :
PTR:
        dc.b     xxx,xxx,xxx,xxx,…
```
サンプル・プログラム

P.238　SMPL28

$FF45 dup(FILENO) 1.0x/2.0x

引数

word　　FILENO ；複写もとのファイル・ハンドル

返り値

D0.L　　複写された新しいファイル・ハンドル．負の数ならエラーコード．

機能

FILENOのファイル・ハンドルを新しいファイル・ハンドルに複写します．$FF2F dup0, $FF46 dup2などの強制複写とは，複写先のハンドルを自動的に割り当てる点で異なります．

コール

```
	move.w	#FILENO,-(SP)
	dc.w	$FF45
	addq.l	#2,SP
```

参照

$FF2F　dup0(FILENO,NEWNO)
$FF46　dup2(FILENO,NEWNO)

サンプル・プログラム

P.241　SMPL30

$FF46 dup2(FILENO,NEWNO) 1.0x/2.0x

引数

word　　FILENO ；複写もとのファイル・ハンドル
word　　NEWNO ；複写先のファイル・ハンドル．

返り値

D0.L　　負の数ならエラーコード．

機能

FILENOのファイル・ハンドルをNEWNOのファイル・ハンドルに強制複写します．NEWNOがオープンされている場合は自動的にクローズしてから複写します．

コール

```
	move.w	#NEWNO,-(SP)
	move.w	#FILENO,-(SP)
	dc.w	$FF46
	addq.l	#8,SP
```

参照

$FF2F　dup0(FILENO,NEWNO)
$FF45　dup(FILENO)

サンプル・プログラム

P.241　SMPL30

$FF47 curdir(DRIVE,PATHBF) 1.0x/2.0x

引数

word　DRIVE　; カレント・ドライブ＝0, A＝1, B＝2, …
long　PATHBUF　; パス・リストが返るバッファを指すポインタ.

返り値

D0.L　負の数の場合はエラーコード

機能

指定したドライブ上のカレント・ディレクトリのパス・リストをPATHBUFの指すバッファに返します.

ディレクトリ構造を深く構築してある場合のことを考えて，バッファは充分大きくとっておくことが望ましいでしょう.

Human1.0xでは/で区切られたパス・リストが返ってきましたが，Human2.0xでは¥で区切らたパス・リストが返ってきます.

コール

```
        pea     PATHBUF
        move.w  #DRIVE,-(SP)
        dc.w    $FF47
        addq.l  #6,SP
                :
PATHBUF:
        ds.b    65
```

参照

$FF3B　chdir(DRIVE)

サンプル・プログラム

P.230　SMPL24

$FF48 malloc[LEN] 1.0x/2.0x

引数

long LEN ; 確保するメモリのバイト数.

返り値

D0.L 確保したメモリ・ブロックへのポインタ (メモリ管理テーブル+$10)
指定したバイト数だけ確保できない場合は, $81000000+最大バイト数が返ります.
完全に確保できない場合は, $8200000xが返ります.

機能

メモリをLENバイトだけ確保します.

プロセスが起動された直後の状態では, 空いているすべてのメモリがそのプロセスのために確保されています. $FF4A setblockでそのプロセスに必要なだけのサイズに変更してから, あらためてこのファンクションで別の用途のためにメモリを確保することになります.

メモリは原則として下位方向から順に割り当てられ, その先頭は必ず下位4ビットが0(つまり, $xxxxx0)のアドレスから始まります. 実際に確保されるメモリのサイズはLENバイト+メモリ管理ポインタ16バイトです.

コール

```
move.l  #LEN,-(SP)
dc.w    $FF48
addq.l  #4,SP
```

参照

$FF49	mfree(MEMPTR)	
$FF4A	setblock(MEMPTR, NEWLEN)	
$FF4B	exec(MD, FIL, P1, P2)	
$FF7D	malloc2(MD, LEN)	2.0x
$FF7E	mfree2(MEMPTR)	2.0x
$FF7F	suballoc(THREAD, MEMPTR, ALLLEN, LEN)	2.0x

サンプル・プログラム

P.219 SMPL15

$FF49 mfree(MEMPTR) 1.0x/2.0x

引数

long　　MEMPTR　；開放するメモリ・ブロックへのポインタ（メモリ管理テーブル＋$10）

返り値

D0.L　　負の数の場合はエラーコード

機能

MEMPTRで指定したメモリ・ブロックを開放します．MEMPTRには$FF48 mallocで得られたポインタを指定します．MEMPTRが0の場合は，そのプロセスに確保されているメモリをすべて開放します．

コール

```
move.l  #MEMPTR, -(SP)
dc.w    $FF49
addq.l  #4, SP
```

参照

$FF48	malloc(LEN)	
$FF4A	setblock(MEMPTR, NEWLEN)	
$FF4B	exec(MD, FIL, P1, P2)	
$FF7D	malloc2(MD, LEN)	2.0x
$FF7E	mfree2(MEMPTR)	2.0x
$FF7F	suballoc(THREAD, MEMPTR, ALLLEN, LEN)	2.0x

サンプル・プログラム

P.219　SMPL15

$FF4A setblock(MEMPTR,NEWLEN) 1.0x/2.0x

引数

long　　MEMPTR　；大きさを変更するメモリ・ブロックへのポインタ（メモリ管理テーブル＋$10）

long　　NEWLEN　；変更後のバイト数．

返り値

D0.L　　負の数の場合はエラーコード．
指定したバイト数に変更できない場合は，$81000000＋最大バイト数が返ります．
完全に変更できない場合は，$8200000xが返ります．

機能

MEMPTRで指定したメモリ・ブロックの大きさを変更します．MEMPTRには$FF48 mallocで得られたポインタを指定します．プロセス起動直後，そのプロセスに割り当てられているメモリ・ブロックを操作する場合はA0＋$10を指定します．

NEWLENは下位24ビットのみ有効です．

コール

```
move.l  #NEWLEN, -(SP)
move.l  #MEMPTR, -(SP)
```

```
        dc.w     $FF4A
        addq.l   #8, SP
```

参照

$FF48　malloc(LEN)
$FF49　mfree(MEMPTR)
$FF4B　exec(MD, FIL, P1, P2)
$FF7D　malloc2(MD, LEN)　2.0x
$FF7E　mfree2(MEMPTR)　2.0x
$FF7F　suballoc(THREAD, MEMPTR, ALLLEN, LEN)　2.0x

サンプル・プログラム

P.219　SMPL15

$FF4B exec[MD,???]　1.0x

引数

word　MD　；このファンクションの動作モードを指定します．

その他のパラメータは，指定したモードによって意味や数が違ってきます．

返り値

★ MDが0の場合

D0.L　正の数の場合はプロセス終了コード．負の数の場合はエラーコード

★ MDが1の場合

D0.L　ロードしたプログラムの実行アドレス．負の数の場合はエラーコード．
A0.L　PSP－$10（メモリ管理テーブル）へのポインタ
A1.L　読み込んだプロセスに確保された最終アドレスへのポインタ
A2.L　コマンドライン文字列を指すポインタ
A3.L　環境ポインタ
A4.L　実行アドレス

★ MDが2の場合

D0.L　負の数の場合はエラーコード

★ MDが3の場合

D0.L　プログラムの長さ．負の数の場合はエラーコード

★ MDが4の場合

D0.L　正の数の場合はプロセス終了コード．負の数の場合はエラーコード

機能

★ MDが0の場合

exec(0, FIL, P1, P2)

long　FIL　；起動するファイルの名前を指すポインタ．上位8ビットはローディングモード（0, 1, 2, 3）．
long　P1　；コマンドライン文字列[*1]を指すポインタ
long　P2　；環境テーブル[*2]を指すポインタ

P1のコマンドラインと，P2の環境ポインタを指定してFILのプログラムをロードし，実行

します．P2の環境ポインタとして$00000000を指定すると，自分と同じ環境になります．
ロードしたプログラムには，PSPの各パラメータの他に，以下の値が渡されます．

A0.L　PSP－$10（メモリ管理テーブル）を指すポインタ
A1.L　読み込んだプロセスに確保された最終アドレスを指すポインタ
A2.L　コマンドライン文字列を指すポインタ
A3.L　環境ポインタ
A4.L　実行アドレス

プログラム実行後はD1～D7/A0～A6は壊れます．

★ MDが1の場合

```
exec(1, FIL, P1, P2)
long  FIL    ；起動するファイルの名前を指すポインタ．上位8ビットはローディン
               グモード（0, 1, 2, 3）．
long  P1     ；コマンドライン文字列を指すポインタ
long  P2     ；環境テーブルを指すポインタ
```

P1のコマンドラインとP2の環境ポインタを指定して，FILのプログラムをロードします．実行はしません．P2の環境ポインタとして$00000000を指定すると，自分と同じ環境になります．
ロードしたプログラムには，PSPの各パラメータの他に，以下の値が渡されます．

A0.L　PSP－$10（メモリ管理テーブル）を指すポインタ
A1.L　読み込んだプロセスに確保された最終アドレスを指すポインタ
A2.L　コマンドラインを指すポインタ
A3.L　環境ポインタ
A4.L　実行アドレス

★ MDが2の場合

```
exec(2, FIL, P1, P2)
long  FIL    ；起動するファイルの名前を指すポインタ．
long  P1     ；コマンドライン文字列が返るバッファを指すポインタ
long  P2     ；環境テーブルを指すポインタ
```

P2の環境に設定されているPathをサーチして，FILのコマンド行をファイル名（ドライブ名，パス名つき）とコマンドラインに分けて，FILとP1のでポイントされる各バッファにセットします．FILのバッファは90バイト以上，P1のバッファは256バイト以上必要です．P2の環境ポインタとして$00000000を指定すると，自分と同じ環境になります．
このファンクション実行後，MD＝0や1のexecファンクションを実行すると，設定しておいたパス・リストにしたがってプログラムをロード/実行できます．

★ MDが3の場合

```
exec(3, FIL, P1, P2)
long  FII    ；起動するファイルの名前を指すポインタ．上位8ビットはローディン
               グモード*1（0, 1, 2, 3）．
long  P1     ；ロード・アドレス
long  P2     ；リミット・アドレス
```

P1のロード・アドレスとP2のリミット・アドレスを指定してFILのプログラムをロードします.

★ MDが4の場合

exec (4, FIL, P1, P2)

```
long  FIL       ; 実行アドレス
long  P1        ; コマンドライン*2文字列を指すポインタ
long  P2        ; 環境テーブル*3を指すポインタ
```

FILは実行アドレスを示し，MD=1でロードした後のプログラムを実行します.

プログラム実行後はD1～D7/A0～A6は壊れます.

プロセス起動直後に新しいプロセスをロード/実行する場合は，あらかじめ$FF4A setblockを使って不用なメモリ領域を開放しておく必要があります.

*1:ローディング

0：拡張子に従いロードする.

1：.Rファイルとしてロードする

2：.Zファイルとしてロードする

3：.Xファイルとしてロードする

*2:コマンドライン文字列の構造

コマンドライン文字列は基本的にNULLコードを行末コードとするASCII文字列ですが，先頭にはNULLコードを除いた文字数が1バイト必要です.

例) dc.b 10,'0123456789',0

*3:環境テーブルの構造

```
        dc.l    環境テーブルの大きさ（この4バイトも含む）
        dc.b    '(環境変数名1)'
        dc.b    '='
        dc.b    '(環境変数1の内容)',0
                :
        dc.b    '(環境変数名n)'
        dc.b    '='
        dc.b    '(環境変数nの内容)',0
        dc.b    0                  * 環境内容の終わり
        dc.b    ??,...             * 環境テーブルの終わりまで無意味なデータ
```

コール

(例はMDが1の場合)

```
        clr.l   -(SP)
        pea     P1
        pea     FIL
```

```
        move.w  #1,-(SP)
        dc.w    $FF4B
        lea     14(SP),SP
                :
P1:
        dc.b    10      * コマンドラインの文字数
        dc.b    'sample31b.s'0
FIL:
        dc.b    'A:¥AS¥AS.X',0
```

参照

$FF48　malloc(LEN)

$FF49　mfree(MEMPTR)

$FF4A　setblock(MEMPTR, NEWLEN)

サンプル・プログラム

P.242　SMPL31

Human68Kのハングアップ・パターン

Human68KはCPUのエラーを検出して，プログラム実行を中止する機能があります．これは，68000CPUのエラーチェック機能を生かして実現しています．そのため，機械語で作成したプログラムにエラーがあっても再起動しなくて済む場合がほとんどです．

このエラーチェックに引っ掛からないエラーが起こるとハングアップ(普通のパソコンのように)してしまいますが，このような状態になるにはいくつかのパターンがあります．

1：割り込みルーチン内部でエラーが発生した場合

エラーを表示するときに次の割り込みが掛かってしまい，しばらくするとスタックがふっ飛んでしまう．

2：エラー表示に使用するプログラムを破壊した場合

エラーを表示するときにエラーが発生するので，しばらくするとスタックがふっ飛ぶ．

3：エラー処理終了後のジャンプ先を破壊した場合

エラーに対して中止を選んでもすぐにエラーが発生する．

4：ぐちゃぐちゃになった場合

4:はどうしようもありませんが，1:や2:はプログラム作成時に注意しておくと，再起動の回数が減ります．

$FF4C exit2[CODE] 1.0x/2.0x

引数

word　　CODE　　；プロセス終了コード．

返り値

なし（戻ってこない）．

機能

CODEのプロセス終了コードを持って，プロセスを非常駐終了します．終了コードを指定する必要がなく，単純にプロセスを終了する場合は，$FF00 exitファンクションを使います．

現在オープンしているファイルは，子プロセスがオープンしたファイルも含めてすべてクローズされます．

コール

```
move.w  #CODE, -(SP)
dc.w    $FF4C
```

参照

$FF00　exit()
$FF31　keeppr(SIZE, CODE)
$FF4D　wait()

サンプル・プログラム

P.242　SMPL31

$FF4D wait[ ] 1.0x/2.0x

引数

なし．

返り値

D0.L　　プロセス終了コード．

機能

プロセス終了コードを返します．

コール

```
dc.w    $FF4D
```

参照

$FF00　exit()
$FF31　keeppr(SIZE, CODE)
$FF4B　exec(MD, FIL, P1, P2)
$FF4C　exit2()

サンプル・プログラム

P.242　SMPL31

$FF4E files(FILBUF,NAMEPTR,ATR)　1.0x/2.0x

引数

```
long    FILBUF   ；結果が返るバッファを指すポインタ
long    NAMEPTR  ；サーチするファイル・ネームを指すポインタ
word    ATR      ；サーチするファイルの属性
```

返り値

```
D0.L     負の数の場合はエラーコード
(FILBUF+0).b    ATR      *
(FILBUF+1).b    DRIVENO  *
(FILBUF+2).w    DIRCLS   *
(FILBUF+4).w    DIRFAT   *
(FILBUF+6).w    DIRSEC   *
(FILBUF+8).w    DIRPOS   *
(FILBUF+10).b
       :        FILNAME  *   ワイルドカード展開済み
(FILBUF+17).b     (8バイト．8バイト以下の場合はスペースで埋める)
(FILBUF+18).b
       :        EXT      *   ワイルドカード展開済み
(FILBUF+20).b     (3バイト．3バイト以下の場合はスペースで埋める)
(FILBUF+21).b   ATR
(FILBUF+22).w   TIME     ;$FF2C gettimeと同一フォーマット
(FILBUF+24).w   DATE     ;$FF2B setdateと同一フォーマット
(FILBUF+26).l   FILELEN
(FILBUF+30).b
       :        PACKEDNAME ;
(FILBUF+52).b     (18+1"."+3+1$00バイト)
```

* OS 内部で使用．書き変えると$FF4F nfilesファンクションが実行できなくなります．

機能

ATRで属性を，NAMEPTRでファイル・ネームを指定して，該当するファイルのうち，最初に発見したものの情報をFILBUFの指すバッファに返します．同じ条件に該当する他のファイルをサーチする場合は$FF4F nfilesを使います．バッファは53バイト必要です．

ファイル・ネームの指定にはワイルドカード（"*","?"）が使えます．

ATRの各ビットの意味は次のとおりです．複数のビットを立てた場合，立っているビットで示される属性のうち，どれかに該当するファイルがサーチされます．

```
bit5    通常のファイル
bit4    ディレクトリ
bit3    ボリュームID
bit2    システム・ファイル
bit1    隠しファイル
bit0    読み込み専用
```

指定した条件に該当するファイルが存在しない場合は，エラーコード－2がD0に返ります．

コール

```
        move.w  #ATR,-(SP)
        pea     NAMEPTR
        pea     FILBUF
        dc.w    $FF4E
        lea     10(SP),SP       *ADDQは+8まで
                :
NAMEPTR:
        dc.b    'B:¥SOURCE¥*.S',0
FILBUF:
        ds.b    53
```

参照

$FF4F　nfiles(FILEBUF)

サンプル・プログラム

P.233　SMPL26

RAMDISK.SYSの問題点

Human68Kには標準でRAMディスク・ドライバがついています．このドライバは，非常に使用頻度が高いのですが，次のような問題点があります．

1つめは，データ・エリアがユーザー・エリアにおかれる点です．このため，データが破壊されてもすぐには分からない場合があります．FATやディレクトリの部分が破壊された場合はシステムが検出してくれるので破壊されたことが分かりますが，データ部分の破壊はシステムにとって検出するのが難しく，ほとんどの場合は分かりません．ですから，破壊されたプログラムなどを実行してしまうことがあります．

2つめは，登録時の初期化チェックが不完全である点です．このチェックはFATとディレクトリだけを対象としているため，データ部の破壊を検出できません．よって，データ部が破壊されていても，初期化しない場合があります．また，キャッシュなどを使用したことによって，FATやディレクトリを示すデータがメモリに残っていると，それをRAMディスク・ドライバによるデータと判断して初期化しない場合があります．このような状態では，破壊されたプログラムや無意味なデータを実行してしまう可能性が充分にあります．

3つめは，データ転送が遅い点です．転送部分を逆アセンブルしてみると，スピードを犠牲にして省メモリ(なつかしい…)しています．メガ単位のメモリを持つX68000ならば，1KBぐらいの無駄は誤差範囲です．その程度の無駄で転送スピードが数倍になるなら，スピードをとった方が良いと思うのですが….

転送スピードの問題は実行内容には影響しませんが，「破壊され易いエリアにおかれたデータ部の破壊を検出できない」という点は注意したいところです．

$FF4F nfiles(FILBUF)　　1.0x/2.0x

引数

long　FILBUF　;$FF4E filesファンクションで作成されたバッファを指すポインタ

返り値

D0.L　負の数の場合はエラーコード

```
(FILBUF+0).b    ATR      *
(FILBUF+1).b    DRIVENO  *
(FILBUF+2).w    DIRCLS   *
(FILBUF+4).w    DIRFAT   *
(FILBUF+6).w    DIRSEC   *
(FILBUF+8).w    DIRPOS   *
(FILBUF+10).b
     :          FILNAME  *
(FILBUF+17).b     (8バイト．8バイト以下の場合はスペースで埋める)
(FILBUF+18).b
     :          EXT      *
(FILBUF+20).b     (3バイト．3バイト以下の場合はスペースで埋める)
(FILBUF+21).b   ATR
(FILBUF+22).w   TIME     ;$FF2C gettimeと同一フォーマット
(FILBUF+24).w   DATE     ;$FF2B setdateと同一フォーマット
(FILBUF+26).l   FILELEN
(FILBUF+30).b
     :          PACKEDNAME
(FILBUF+52).b     (18+1"."+3+1$00バイト)
```

* OS 内部で使用．書き変えると$FF4F nfilesファンクションが実行できなくなります．

機能

$FF4E filesファンクションで作成したFILBUFを指定して，同じ条件に該当する次のファイルをサーチして，その情報を返します．

このファンクションは$FF4E FILESファンクション実行後に使ってください．

指定した条件に該当するファイルがこれ以上存在しない場合は，エラーコード－2がD0に返ります．

コール

```
        pea     FILBUF
        dc.w    $FF4F
        addq.l  #4,SP
                :
FILBUF:
        ds.b    53
```

参照

$FF4E　files(FILBUF,NAMEPTR,ATR)

サンプル・プログラム

P.233　SMPL26

$FF50 setpdb[PDBADR]　1.0x/2.0x

引数

long　PDBADR　；管理を移すプロセスID

返り値

D0.L　管理が移る前のプロセスID

機能

PDBADRで示すプロセスに管理を移します．PDBADRは$FF51 getpdbファンクションで与えられたプロセスID（PSP＋$10）でなければいけません．

このコールはジャンプやコールを行なうものではなく，現在実行中のプログラムがどのプロセスとして管理されているかをHumanに通知するものです．プロセス管理に混乱を生じる場合がありますから，使用に際しては充分注意してください．

コール

```
        move.l  PDBADR,-(SP)
        dc.w    $FF50
        addq.l  #4,SP
                :
PDBADR:
        dc.l    xxxxxxxx        * getpdbで得たプロセスIDの待避ワーク
```

参照

$FF26　pspset(PSPADR)
$FF51　getpdb()

サンプル・プログラム

P.219　SMPL15

$FF51 getpdb[ ]　1.0x/2.0x

引数

なし．

返り値

D0.L　現在のプロセスID

機能

現在のプロセスを表わすプロセスID（PSP＋$10）を求めます．

コール

```
        dc.w    $FF51
        move.l  d0,PDBADR
                :
PDBADR:
        ds.l    1
```

参照

$FF26　pspset(PSPADR)
$FF50　setpdb(PDBADR)

サンプル・プログラム

P.219　SMPL15

$FF52 setenv(SETNAME,ENV,SETLINE)　　1.0x/2.0x

引数

long　SETNAME ; 環境変数の名前を指すポインタ
long　ENV　　 ; 変数を定義する環境を指すポインタ
long　SETLINE ; 環境変数に代入する文字列を指すポインタ

返り値

D0.L　負の数の場合はエラーコード

機能

ENVで指定した環境に，SETNAMEの環境変数をSETLINEの内容で定義します．
ENVを$00000000にすると，現在のプロセスの環境に定義します．
SETLINEを$00000000にすると，SETNAMEの変数を消去します．
環境変数の内容は255バイト以内です．

コール

```
        pea     SETLINE
        clr.l   -(SP)
        pea     SETNAME
        dc.w    $FF52
        lea     12(SP),SP
                :
SETNAME:
        dc.b    'temp',0
SETLINE:
        dc.b    'C:',0
```

参照

$FF53　getenv(GETNAME,ENV,GETBUF)

サンプル・プログラム

P.244　SMPL32

$FF53 getenv(GETNAME,ENV,GETBUF)　1.0x/2.0x

引数

- GETNAME ; 環境変数の名前を指すポインタ
- ENV ; 内容を読み出す環境変数が存在する環境を指すポインタ
- GETBUF ; 環境変数の内容が返るバッファを指すポインタ

返り値

負の数の場合はエラーコード

機能

ENVで指定した環境の，GETNAMEの環境変数の内容をGETLINEの指すバッファに読み出します。

ENVを$00000000にすると，現在のプロセスの環境が指定されます．

バッファは256バイト必要です．

コール

```
        pea     GETBUF
        clr.l   -(SP)
        pea     GETNAME
        dc.w    $FF53
        lea     12(SP),SP
                :
GETNAME:
        dc.b    'temp',0
GETBUF:
        ds.b    256
```

参照

$FF52 setenv(SETNAME,ENV,SETLINE)

サンプル・プログラム

P.244 SMPL32

$FF54 verifyg( ) 1.0x/2.0x

引数

なし.

返り値

D0.L　ベリファイフラグの設定状況

　　＝0　ベリファイしない

　　＝1　ベリファイする

機能

ベリファイフラグの設定状況を返します.

コール

```
        dc.w    $FF54
```

参照

$FF2E　verify(FLG)

サンプル・プログラム

P.246　SMPL33

$FF56 rename(OLD,NEW) 1.0x/2.0x

引数

long　OLD　；リネームするファイル・ネームを指すポインタ

long　NEW　；リネーム後のファイル・ネームを指すポインタ

返り値

D0.L　負の数の場合はエラーコード

機能

OLDで示されるファイルのファイル・ネームをNEWに変更します.

OLDとNEWの間でパスが異なっている場合は，OLDのファイルをNEWに変更した上で，NEWのパスに移動させることに注意してください.

OLDとNEWのドライブが異なる場合はエラーとなります.

コール

```
        pea     NEW
        pea     OLD
        dc.w    $FF56
        addq.l  #8,SP
                :
OLD:
        dc.b    'SAMPLE.S',0
NEW:
        dc.b    'SAMPLE56.S',0
```

サンプル・プログラム

P.247　SMPL34

$FF57 filedate(FILENO,DATETIME)　　1.0x/2.0x

引数

word　FILENO　；ファイル・ハンドル
long　DATETIME　；設定する日付/時刻

返り値

D0.L　DATETIME＝0の場合は読み出したファイルの日時.

b31←――――――――――――→b0

YYYYYYM MMMDDDDD hhhhhmmm mmmsssss

bit25～31(YYYYYYY)	0～199年(＋1980年)
bit21～24(MMMM)	1～12月
bit16～20(DDDDD)	1～31日
bit11～15(hhhhh)	0～23時
bit 5～10(mmmmmm)	0～59分
bit 0～ 4(ssss)	0～29秒

上位16ビットが$FFFFならエラーコード（「負の数」でエラーコードであると判断できません）.

機能

FILENOのファイル・ハンドルで示すファイルの日付/時間を読み出したり，設定したりします.

DATETIMEが$00000000の場合は読み出しを行ないます.

コール

```
move.l  #DATETIME,-(SP)
move.w  #FILENO,-(SP)
dc.w    $FF57
addq.l  #6,SP
```

参照

$FF3D　open(NAMPETR,MOME)

サンプル・プログラム

P.222　SMPL17

$FFF0 retshell( ) 仮 1.0x/2.0x

引数

なし

返り値

なし

機能

このコールのベクタとして，シェルへのリターン・アドレスが指定されています．

直接呼び出した場合，スタックや各種ベクタの復帰などが行なわれないままでシェルに戻ってしまうことになり，問題が発生します．したがって，ユーザーがファンクションコールとして使ってはいけません．

$FFF1 ctrlcabort( ) 仮 1.0x/2.0x

引数

なし

返り値

なし

機能

このコールのベクタとして，コントロールCが押された場合のアボート・アドレスが指定されています．

ユーザーがファンクションコールとして利用できませんが，ベクタを書き換えておくことでコントロールCでアボートした場合の処理を変更できます．

このアボートベクタはPSP内に保存されていますから，プロセス内で書き換えたとしてもプロセス終了時にはもとの値に復帰します．

$FFF2 errabort( ) 仮 1.0x/2.0x

引数

なし

返り値

なし

機能

このコールのベクタとして，エラーが発生したなどの原因でプロセスをアボートする場合のアドレスが指定されています．

ユーザーがファンクションコールとして利用できませんが，ベクタを書き換えておくことでエラーアボートの処理を変更できます．

このアボートベクタはPSP内に保存されていますから，プロセス内で書き換えたとしてもプロセス終了時にはもとの値に復帰します．

$FFF3 diskred(ADR,DRIVE,SECT,SECTLEN) 1.0x/2.0x

引数

```
long    ADR     ；ディスクから読み出した結果が入るバッファを指すポインタ
word    DRIVE   ；カレント・ドライブ＝0, A＝1, B＝2…
word    SECT    ；読みだし開始セクタ（0～）
word    SECTLEN ；読み出すセクタ数（1～）
```

返り値

なし.

機能

ディスクの直接読み出しを行ないます．バッファは１セクタに対して1024バイトずつ必要です（Ver2.0xで１セクタ1024バイト以上のデバイスをアクセスする場合はバッファも1024バイト以上必要です）．

コール

```
        move.w  #SECTLEN,-(SP)
        move.w  #SECT,-(SP)
        move.w  #DRIVE,-(SP)
        pea     ADR
        dc.w    $FFF3
        lea     10(SP),SP
                :
ADR:
        ds.b    1024*n
```

参照

$FFF4 diskwrt(ADR, DRIVE, SECT, SECTLEN)

サンプル・プログラム

P.258 SMPL41

$FFF4 diskwrt(ADR,DRIVE,SECT,SECTLEN) 1.0x/2.0x

引数

long　　ADR　　；ディスクに書き込むデータの存在するバッファを指すポインタ
word　　DRIVE　；カレント・ドライブ＝0, A＝1, B＝2…
word　　SECT　　；書き込み開始セクタ（０～）
word　　SECTLEN；書き込むセクタ数（１～）

返り値

なし．

機能

ディスクの直接書き込みを行ないます．バッファは１セクタに対して1024バイトずつ必要です(Humanr2.0x で１セクタ1024バイト以上のデバイスをアクセスする場合はバッファも1024バイト以上必要です)．

コール

```
        move.w  #SECTLEN,-(SP)
        move.w  #SECT,-(SP)
        move.w  #DRIVE,-(SP)
        pea     ADR
        dc.w    $FFF4
        lea     10(SP),SP
                :
ADR:
        dc.b    xx,xx,xx,xx,…
```

参照

$FFF3　diskred(ADR,DRIVE,SECT,SECTLEN)

サンプル・プログラム

P.258　SMPL41

$FF33 breakck(FLG) 2.0x

引数

word FLG ;ブレイクフラグ

=0 指定のファンクションコールのみチェックする

=1 すべてのファンクションコールでチェックする

=2 すべてのファンクションコールでブレイクを無視する

=−1 設定状況を見るだけ

返り値

D0.L ブレイクフラグの設定状況

=0 指定のファンクションコールでのみブレイクチェックする

=1 すべてのファンクションコールでブレイクチェックする

=2 すべてのファンクションコールでブレイクを無視する

機能

ブレイクフラグを設定/参照します．Human2.0xになってブレイクを無視する設定が追加されました．

コール

```
move.w  #FLG,-(SP)
dc.w    $FF33
addq.l  #2,SP
```

サンプル・プログラム

P.226 SMPL20

X68000起動時の特殊キー

X68000の電源を入れて起動する際に，特定のキーを押しながら起動することによって，さまざまな状態でX68000を使い始めることができます．案外知られていないものもありますので，ここでまとめておきましょう．

ハードウェアやIPL ROMによって実現されている機能は，Human以外のアプリケーション(OS9など)でも使うことができます．

ハード/IPLで実現されている機能		Humanによって実現されている機能	
キー	機能	キー	機能
OPT.1	メモリ・スイッチの内容に関係なく，スタンダードの起動順序(FD→HD→ROM→RAM)で起動します．	SHIFT	RAMディスク，SRAMディスクの内容を初期化しながら起動します．
OPT.2	メモリ・スイッチの内容に関係なく，ROMデバッガを起動してからブートします．	HELP	ハードディスクを分割して使っている場合，どのエリアから起動するかを選択するメニューを表示させる．
XF1 〜 XF5	キーボードのランプの明るさを指定します．XF1が一番明るく，XF5がいちばん暗くなります．		

$FF3D open(NAMEPTR,MODE) 2.0x

引数

long　NAMEPTR ; ファイル・ネームを指すポインタ
word　MODE　 ; ファイルをオープンするモード

返り値

D0.L　ファイル・ハンドル．負の数の場合はエラーコード．

機能

★ MODEが$0x0の場合
　読み込みオープン
★ MODEが$0x1の場合
　書き込みオープン
★ MODEが$0x2の場合
　読み込み/書き込みオープン
★ MODEが$1x0の場合
　読み込みオープン（辞書用の特殊ハンドラを返す．ユーザーは使用禁止）
★ MODEが$1x1の場合
　書き込みオープン（辞書用の特殊ハンドラを返す．ユーザーは使用禁止）
★ MODEが$1x2の場合
　読み込み/書き込みオープン（辞書用の特殊ハンドラを返す．ユーザーは使用禁止）

MODEのbit6～bit4ではファイルのシェアリングモードを指定します．なお，CONFIG.SYSで"SHARE="を指定していない場合，シェアリングモードの指定はできず，オープンしたファイルの管理はHuman1.0xと同様（シェアリングモード『不可なし』と同等）になります．

表11

bit6	bit5	bit4	シェアリングモード
0	0	0	コンパチブルモード
0	0	1	読み込み/書き込み不可
0	1	0	書き込み不可
0	1	1	読み込み不可
1	0	0	不可なし

☆コンパチブルモード

コンパチブルモードでオープンしたファイルは，読み出しオープンについては無制限にオープンできます．

次のような場合，共有違反となり，エラーを返します．

- すでにコンパチブルモードでオープンされているファイル（読み込み，書き込み，読み込み/書き込みモードにかかわらず）を書き込み，読み込み/書き込みオープンしようとした場合．
- すでにコンパチブルモードで書き込み，読み込み/書き込みオープンされているファイルを，再びコンパチブルモードでオープン（読み込み，書き込み，読み込み/書き込みモードにかかわらず）しようとした場合．

読み込み／書き込み不可

このモードでオープン中のファイルは，クローズするまではいかなるモードでもオープンできません．

書き込み不可

このモードでオープン中のファイルは，クローズするまではコンパチブルモードでオープンしたり，書き込みアクセスしたり読み込み/書き込みアクセスしたりできません．コンパチブルモード以外でオープンして，読み込みアクセスすることは可能です．

読み込み不可

このモードでオープン中のファイルは，クローズするまではコンパチブルモードでオープンしたり，読みだしアクセスや読み込み/書き込みアクセスしたりできません．コンパチブルモード以外でオープンして，書き込みアクセスすることは可能です．

不可なし

このモードでオープン中のファイルは，クローズするまではコンパチブルモードでオープンできませんが，それ以外のモードでオープンすれば読み込み，書き込み，読み込み/書き込みアクセスともに可能です．

この他はHuman1.0xと同様です．

コール

```
        move.w  #MODE,-(SP)
        pea     NAMEPTR
        dc.w    $FF3D
        addq.l  #6,SP
                :
NAMEPTR:
        dc.b    'A:DATA¥DATA_P0',0
```

参照

$FF3C　create(NAMEPTR,MODE)
$FF3E　close(FILENO)
$FF5A　creattmp(NAMEPTR,MODE)　　2.0x
$FF5B　create2(NAMEPTR,MODE)　　2.0x

サンプル・プログラム

P.207　SMPL9

$FF4B exec(MDL * 256+MD,???) 2.0x

引数

word MDL*256＋MD ；このファンクションの動作モードを指定します．MDが0, 1, 3の場合，MODはモジュール番号の指定となります．

その他のパラメータは，指定したモードによって意味や数が違ってきます．

返り値

★ MD が 0 の場合

D0.L 正の数の場合はプロセス終了コード．負の数の場合はエラーコード

★ MD が 1 の場合

D0.L ロードしたプログラムの実行アドレス．負の数の場合はエラーコード．

A0.L PSP－$10（メモリ管理テーブル）へのポインタ

A1.L 読み込んだプロセスに確保された最終アドレスへのポインタ

A2.L コマンドライン文字列を指すポインタ

A3.L 環境ポインタ

A4.L 実行アドレス

★ MD が 2 の場合

D0.L 負の数の場合はエラーコード

★ MD が 3 の場合

D0.L プログラムの長さ．負の数の場合はエラーコード

★ MD が 4 の場合

D0.L 正の数の場合はプロセス終了コード．負の数の場合はエラーコード

★ MD が 5 の場合

D0.L 指定したXファイルのモジュール番号×256．負の数の場合はエラーコード

機能

★ MD が 0 の場合

exec(MOD*256＋0, FIL, P1, P2)

long FIL ；起動するファイルの名前を指すポインタ．上位 8 ビットはローディングモード（0, 1, 2, 3）．

long P1 ；コマンドライン文字列を指すポインタ

long P2 ；環境テーブルを指すポインタ

P1のコマンドラインと，P2の環境ポインタを指定してFILのプログラムをロードし，実行します．P2の環境ポインタとして$00000000を指定すると，自分と同じ環境になります．

MODは通常 0 ですが，オーバレイXファイルの中から他のモジュールをロード/実行する場合は，MODでモジュール番号を指定します．また，この場合，ファイル・ネームとして指定するのは起動したいモジュールを含んだオーバレイXファイルの名前である点に注意してください．

ロードしたプログラムには，PSPの各パラメータの他に，以下の値が渡されます．

A0.L PSP－$10（メモリ管理テーブル）を指すポインタ

A1.L 読み込んだプロセスに確保された最終アドレスを指すポインタ

A2.L コマンドライン文字列を指すポインタ

A3.L 環境ポインタ

A4.L　　実行アドレス

プログラム実行後はD1～D7/A0～A6は壊れます.

★ MDが1の場合

exec(MOD*256+1, FIL, P1, P2)

long　FIL　　；起動するファイルの名前を指すポインタ．上位8ビットはローディングモード（0, 1, 2, 3).

long　P1　　；コマンドライン文字列を指すポインタ

long　P2　　；環境テーブルを指すポインタ

P1のコマンドラインとP2の環境ポインタを指定して，FILのプログラムをロードします．実行はしません．P2の環境ポインタとして$00000000を指定すると，自分と同じ環境になります.

MODは通常0ですが，オーバレイXファイルの中から他のモジュールをロードする場合は，MODでモジュール番号を指定します．また，この場合，ファイル・ネームとして指定するのは起動したいモジュールを含んだオーバレイXファイルの名前である点に注意してください.

ロードしたプログラムには，PSPの各パラメータの他に，以下の値が渡されます.

A0.L　　PSP－$10(メモリ管理テーブル)を指すポインタ

A1.L　　読み込んだプロセスに確保された最終アドレスを指すポインタ

A2.L　　コマンドラインを指すポインタ

A3.L　　環境ポインタ

A4.L　　実行アドレス

★MDが2の場合

exec(2, FIL, P1, P2)

long　FIL　　；起動するファイルの名前を指すポインタ.

long　P1　　；コマンドライン文字列が返るバッファを指すポインタ

long　P2　　；環境テーブルを指すポインタ

P2の環境に設定されているPathをサーチして，FILのコマンド行をファイル名(ドライブ名，パス名つき)とコマンドラインに分けて，FILとP1でポイントされる各バッファにセットします．FILのバッファは90バイト以上，P1のバッファは256バイト以上必要です．P2の環境ポインタとして$00000000を指定すると，自分と同じ環境になります.

このファンクション実行後，MD＝0や1のexecファンクションを実行すると，設定しておいたパス・リストにしたがってプログラムをロード/実行できます.

★ MDが3の場合

exec(MOD*256+3, FIL, P1, P2)

long　FIL　　；起動するファイルの名前を指すポインタ．上位8ビットはローディングモード（0, 1, 2, 3).

long　P1　　；ロード・アドレス

long　P2　　；リミット・アドレス

P1のロード・アドレスとP2のリミット・アドレスを指定してFILのプログラムをロードします．MODは通常0ですが，オーバレイXファイルの中から他のモジュールをロードする場合は，MODでモジュール番号を指定します．また，この場合，ファイル・ネームとして指定するのは起動したいモジュールを含んだオーバレイXファイルの名前である点に注意してください.

★ MDが4の場合

exec (4, FIL, P1, P2)

long　FIL　　；実行アドレス

long　P1　　；コマンドライン文字列を指すポインタ

long　P2　　；環境テーブルを指すポインタ

FILは実行アドレスを示し，MD＝1でロードした後のプログラムを実行します．

プログラム実行後はD1～D7/A0～A6は壊れます．

★MDが5の場合

exec (5, FIL, P1)

long　FIL　　；オーバレイXファイル名を指すポインタ

long　P1　　；サーチするモジュールの名前を指すポインタ

FILで指定したオーバレイXファイルの中からP1で指定したモジュールを捜し，みつかった場合そのモジュール番号*256を返します．

コール

(例はMDが5の場合)

```
        pea     P1
        pea     FIL
        move.w  #$00_05,-(SP)
        dc.w    $FF4B
        lea     10(SP),SP
                :
P1:
        dc.b    'DUMP'0
FIL:
        dc.b    'MENU.X',0
```

参照

$FF48　malloc (LEN)

$FF49　mfree (MEMPTR)

$FF4A　setblock (MEMPTR, NEWLEN)

サンプル・プログラム

P. 242　SMPL31

P. 472　ABD. X

$FF55 common[MD,NAME,???] 仮　　2.0x

引数

word　MD　；このファンクションの動作モードを指定します．
long　NAME　；アクセスするデータ・ブロック名を指すポインタ（12バイト以内）．
その他のパラメータは，指定したモードによって意味や数が違ってきます．

返り値

★ MD が0の場合
D0.L　データ・ブロックの長さ．負の数の場合はエラー．

★ MD が1の場合
D0.L　実際に読み出したデータのバイト数．負の数の場合はエラー．

★ MD が2の場合
D0.L　指定したバッファの内容をブロックに書き込んだ長さ．負の数の場合はエラー．

★ MD がその他の場合
D0.L　負の数の場合はエラー．

機能

CONFIG.SYSの"COMMON="によって設定されたコモンエリアをアクセスするためのファンクションです．"COMMON="でコモンエリアのサイズを指定していない場合は，このファンクションも使えません．

★ MD が0の場合
common (0, NAME)
NAMEで指定した名前の付いたデータ・ブロックの長さをD0に返します．
指定した名前のブロックが存在しない場合はエラーとなります．

★ MD が1の場合
common (1, NAME, OFS, PTR, LEN)
long　OFS　；読みだし開始オフセット
long　PTR　；データを読み込むバッファを指すポインタ
long　LEN　；読み出すバイト数

NAMEで指定した名前の付いたデータ・ブロックの内容を，OFSバイト目から，LENで指定する長さだけ，PTRで指定したバッファに読み出します．D0には実際に読み出したバイト数が返ります．バッファはLENバイト以上必要です．
指定した領域の一部でも他のプロセスによってロックされている場合や，指定した名前のブロックが存在しない場合はエラーとなります．

★ MD が2の場合
common (2, NAME, OFS, PTR, LEN)
long　OFS　；書き込み開始オフセット
long　PTR　；書き込むデータの入ったバッファを指すポインタ
long　LEN　；書き込むバイト数

NAMEで指定した名前の付いたデータ・ブロックの，OFSバイト目から，LENで指定する長さだけ，PTRで指定したバッファの内容を書き込みます．
指定した領域の一部でも他のプロセスによってロックされている場合はエラーとなります．

★ MD が 3 の場合

common (3, NAME, OFS, PSPID, LEN)

long　OFS　　；ロック開始オフセット

long　PSPID　；ロックを行なうプロセス（自分）のID

long　LEN　　；ロックする領域のバイト数

NAMEで指定した名前の付いたデータ・ブロックのOFSからLENバイトの他のプロセスからのアクセスをロックします．

PSPIDには，現在のプロセスID（メモリ管理テーブル＋$10）を指定してください．

自分，または他のプロセスがこのデータ・ブロックをロックしている場合は，ロックを行なうことができません．

★ MD が 4 の場合

common (4, NAME, OFS, PSPID, LEN)

long　OFS　　；ロック解除オフセット

long　PSPID　；ロックを解除するプロセス（自分）のID

long　LEN　　；ロック解除する領域のバイト数

NAMEで指定した名前の付いたデータ・ブロックのロックを解除します．

ロックを解除するためには，そのデータ・ブロックのロック領域を設定した場合と同一の引数，OFS，PSPID，LENを指定する必要があり，違う場合はエラーとなります．

★MDが 5 の場合

common (5, NAME)

NAMEで指定した名前の付いたデータ・ブロックを削除します．

目的のデータ・ブロックが他のプロセスによってロックされている場合，削除できません (自分がロックしているのならば削除も可)．

コール

(例はMDが 1 の場合)

```
        move.l  #LEN, -(SP)
        pea     PTR
        move.l  #OFS, -(SP)
        pea     NAME
        move.w  #1, -(SP)
        dc.w    $FF55
        lea     18(SP),SP
                :
NAME:
        dc.b    'MSG0',0
PTR:
        dc.b    xx,xx,xx,...
```

参照

$FFFD　message (SENDER, RECIEVER, ATR, MESPTR, LEN)　2.0x

サンプル・プログラム

P.472　ABD.X

$FF58 malloc(LEN)

$FF48のmallocとまったく同一

$FF5A creattmp(NAMEPTR,ATR) 仮　　2.0x

引数

long　NAMEPTR　；ファイル・ネームを指すポインタ
word　ATR　　　；ファイルの属性

返り値

D0.L　ファイル・ハンドル．負の数の場合はエラーコード．
(NAMEPTR).b～　作成されたテンポラリ・ファイル・ネーム

機能

テンポラリ・ファイルを作成します．

$FF3Cのcreateと違う点は，'ファイル・ネーム中に特殊文字'?'が使える点です．Humanは'?'を適当な数字に置き換えることによって，既に存在するファイルの名前と重複しないようなファイル・ネームを生成し，その後createと同様な作業を行ないます．

'?'を使わない場合でも，指定したファイル名と同名のファイルが存在した場合，そのファイル名が数字を含んでいれば，数字を＋1することによって重複しないファイル名のファイルを作成します．

既に存在するファイルの名前と重複しない名前を生成できなかった場合，エラーが発生し，ファイルの作成は行ないません．

作成されたテンポラリ・ファイルはシェアリングの対象となります．したがって，その数は，CONFIG.SYSの"SHARE＝"の第１パラメータで指定したファイル数を超えることはできません．それ以上作成しようとした場合，エラーにはならず，作成できる状況（例：他のプロセスがシェアリングを行なっていたファイルをクローズした場合など）になるまで待ってしまいます．

CONFIG.SYSで"SHARE＝"を指定していない場合，このような現象は発生せず，いくらでもテンポラリ・ファイルを作成できます．

ATRの各ビットを立てた場合，以下のようなファイルの属性が指定されます．複数のビットを立てることも可能です．

bit5	通常のファイル
bit4	ディレクトリ
bit3	ボリュームID
bit2	システム・ファイル
bit1	隠しファイル
bit0	読み込み専用ファイル

(その他のビットは意味を持たない)

コール

```
move.w #$20,-(SP)
```

```
        pea     NAMEPTR
        dc.w    $FF5A
        addq    #6,SP
                :
NAMEPTR:
        dc.b    'TMP????',0
```

参照

```
    $FF3C  create(NAMEPTR,MODE)
    $FF3D  open(NAMEPTR,MODE)
    $FF3E  close(FILENO)
    $FF5B  create2(NAMEPTR,MODE)              2.0x
```

サンプル・プログラム

P.248　SMPL35

プログラミングに便利なツール類(Ⅰ)

X68Kのソフトは市販のものはもちろん，PDSとしてパソコン通信で流通しているものや，雑誌に掲載されたものにも素晴らしいものがあります．

そうしたソフトの中で，プログラミングの際に私も重宝しているものをいくつかご紹介します．

● μEMACS 3.9e（移植:OBJECT-X氏）

マルチ・ファイル/マルチ・ウィンドウ対応のスクリーン・エディタです．μEMACSにはいくつかのバージョンが存在しますが，この“3.9e”というのはユーザーがマクロを使って好みの機能をプログラミングし，エディタに組み込むことができるようになっている点で『通好み』と言えます．

アセンブラのソースを書くときにはもちろん，Cのソースを書く場合には絶大な威力を発揮します．日本語禁則機能を付加したOBJECT-X版ならばワープロがわりにも使えます．

このソフトはフリーウェアとして PEKIN などで入手できます（ソース付）．

また，さらに強力な μEMACS 3.10も登場しました．

● undel（作:おおすず氏）

テストランやソースの修正を繰り返している間に，うっかり必要なファイルをデリートしてしまった経験を持つ方も少なくないでしょう．いままでは大変な手間をかけて手作業で復活させるか，諦めるかしかなかったわけですが，このソフトを使うことによって誰でも簡単にファイルを復活できるようになりました．諦めるクチだった私などにとっては救世主のようなソフトです．

このソフトはフリーウェアとして **PEKIN，梁山泊 NET，サンデーネット，電脳倶楽部**（ディスクマガジン）などで入手できます（ソース付）．［P.161に続く］

$FF5B create2(NAMEPTR,MODE) 仮　　2.0x

引数

long　NAMEPTR　；ファイル・ネームを指すポインタ
word　ATR　　；ファイルの属性

返り値

D0.L　ファイル・ハンドル．負の数の場合はエラーコード．

機能

ファイルを作成します．

$FF3Cのcreateと違う点は，指定した名前のファイルが既に存在していた場合，エラーとなりファイルを作成しない点です．

ATRの各ビットを立てた場合，以下のようなファイルの属性が指定されます．複数のビットを立てることも可能です．

bit5	通常のファイル
bit4	ディレクトリ
bit3	ボリュームID
bit2	システム・ファイル
bit1	隠しファイル
bit0	読み込み専用ファイル

(その他のビットは意味を持たない)

コール

```
        move.w  #$20,-(SP)
        pea     NAMEPTR
        dc.w    $FF5A
        addq    #6,SP
                :
NAMEPTR:
        dc.b    'TMP0000',0
```

参照

$FF3C　create(NAMEPTR,MODE)
$FF3D　open(NAMEPTR,MODE)
$FF3E　close(FILENO)
$FF5A　creattmp(NAMEPTR,MODE)　　2.0x

サンプル・プログラム

P.249　SMPL36

$FF5C lock(MD,FNO,OFS,LEN) 仮　　2.0x

引数

word　MD　；このファンクションの動作モードを指定します.
word　FNO　；ファイル・ハンドル
long　OFS　；ロックする領域のオフセット
long　LEN　；ロックする領域の長さ

返り値

D0.L　負の数の場合はエラー.

機能

このファンクションは,CONFIG.SYSの中で"SHARE＝"でロックを行なうファイル数と領域数を設定していなければ使えません.

★ MDが0の場合

FNOで指定されたファイルの, オフセットOFSからLENバイトの領域をロックします.

ロックされた領域は, ロックを行なったプロセス以外のプロセスからは読み出しも書き込みもできなくなります.

★ MDが1の場合

FNOで指定されたファイルの,オフセットOFSからLENバイトの領域のロックを解除します.

ロックを解除できるのはロックを行なったプロセスに限られます, また, OFSとLENはロックしたときと同じ値を指定してください.

コール

```
move.l  #LEN,-(SP)
move.l  #OFS,-(SP)
move.w  #FNO,-(SP)
move.w  #0,-(SP)
dc.w    $FF5C
lea     12(SP),SP
```

参照

$FF3C　create(NAMEPTR,MODE)
$FF3D　open(NAMEPTR,MODE)
$FF3E　close(FILENO)
$FF5A　creattmp(NAMEPTR,MODE)　2.0x
$FF5B　create2(NAMEPTR,MODE)　2.0x

サンプル・プログラム

P.250　SMPL37

$FF5F assign(MD,???) 仮　　2.0x

引数

word　MD　　；このファンクションの動作モードを指定します．

その他のパラメータは，指定したモードによって意味や数が違ってきます．

返り値

★ MDが0の場合

D0.L　アサインインデックス．負の数の場合はエラー

(BUF).b〜　ドライブのアサイン状況

● アサインインデックスが$40だった場合

指定したドライブはノーマルな状態のドライブであり，BUFで指定したバッファにはそのドライブのカレント・ディレクトリのパス名が返ります．

● アサインインデックスが$50だった場合

指定したドライブは仮想ドライブであり，BUFで指定したバッファにはその仮想ドライブに割り付けられているサブディレクトリのパス名が返ります．

● アサインインデックスが$60だった場合

指定したドライブは他のドライブのサブディレクトリに割り付けられており，BUFで指定したバッファにはそのドライブが割り付けられているサブディレクトリのパス名が返ります．

★ MDが1の場合

D0.L　負の数の場合はエラー

★ MDが4の場合

D0.L　負の数の場合はエラー

機能

ドライブをディレクトリとしてアクセスしたり，ディレクトリを仮想ドライブとしてアクセスする場合の設定を行なうファンクションです．

仮想ドライブ名はCONFIG.SYS中の"LASTDRIVE＝"で指定した最終ドライブ名を超えることはできません．

現在，このファンクションには仮想ドライブ関係の機能しか用意されていませんが，Human68k用のLANが登場した場合，拡張される可能性があります．

★ MDが0の場合

assign(0,DRVNAME,BUF)

long　DRVNAME ；ドライブ名を指すポインタ

long　BUF　　；ドライブのアサイン状況が返るバッファを指すポインタ

DRVNAMEで指定するドライブのアサイン状況を見ます．アサインインデックスがD0に，アサイン状況がBUFで指定したバッファに返ります．

★ MDが1の場合

assign(1,DRVNAME,PATHNAME,IDX)

long　DRVNAME ；ドライブ名を指すポインタ

long　PATHNAME；パス名を指すポインタ

word　IDX　　；アサインインデックス

● IDXが$50の場合

DRVNAMEで指定したドライブを,PATHNAMEで指定したパス名でアクセスできるように設定を行ないます.

● IDXが$60の場合

PATHNAMEで指定したパスを,DRVNAMEで指定した仮想ドライブ名でアクセスできるように設定を行ないます.

★ MD が 4 の場合

assign(3, DRVNAME)

long DRVNAME ;ドライブ名を指すポインタ

DRVNAMEで指定するドライブのアサイン状況をデフォルト状態に戻します.

コール

(MDが 1 の場合)

```
        move.w  #$60, -(SP)
        pea     PATHNAME
        pea     DRVNAME
        move.w  #1, -(SP)
        dc.w    $FF5F
        lea     12(SP),SP
                :
PATHNAME:
        dc.b    "a:¥work¥source",0
DRVNAME:
        dc.b    "w:",0
```

参照

$FF34 drvxchg(OLD, NEW)

サンプル・プログラム

P.252 SMPL38

$FF7C getfcb(FNO) 仮 2.0x

引数

word FNO ;ファイル・ハンドル

返り値

D0.L FCBのアドレス．負の数の場合はエラーコード

機能

オープンされているファイルのFCB（ファイル・コントロールブロック）のアドレスを求めます.

FCBは1つのファイルにつき$60バイトずつ用意されています.

コール

```
        move.w  #FNO, -(SP)
        dc.w    $FF7C
        addq.l  #2,SP
```

サンプル・プログラム

P.253 SMPL39

$FF7D malloc2(MD,LEN) 仮 2.0x

引数

word　MD　；メモリの割り当て方法を指定します
long　LEN　；確保するメモリのバイト数

返り値

D0.L　確保したメモリ・ブロックへのポインタ（メモリ管理テーブル＋$10）
　指定したバイト数だけ確保できない場合は，$81000000＋最大バイト数が返ります．
　完全に確保できない場合は，$8200000xが返ります．

機能

メモリをLENバイトだけ確保します．
$FF48　mallocとの違いは，mallocがメモリの下位方向から順にメモリを割り当てていくのみであったのに対し，このファンクションでは上位方向からの割り当てや，最小メモリ・ブロックの割り当てなどが可能になっていることと，スレッドの管理メモリ領域操作に完全対応していることの２点です．

★ MD が０の場合
最も下位にあり，指定されたバイト数だけ確保できるメモリ・ブロックを割り当てます．
$FF48　mallocと同様．

★ MD が１の場合
指定されたバイト数を確保できるだけの最小のメモリ・ブロックを全メモリ中から探し出し，割り当てます．

★ MD が２の場合
メモリの上位方向から必要なバイト数だけ割り当てます．

コール

```
move.l  #LEN,-(SP)
move.w  #MD,-(SP)
dc.w    $FF7D
addq.l  #6,SP
```

参照

$FF48	malloc(LEN)	
$FF49	mfree(MEMPTR)	
$FF4A	setblock(MEMPTR,LEN)	
$FF7E	mfree2(MEMPTR)	2.0x
$FF7F	setmarea(THREAD,MEMPTR,ALLLEN,LEN)	2.0x

サンプル・プログラム

P.255　SMPL40

$FF7E mfree2(MEMPTR) 仮　　2.0x

引数

long　　MEMPTR　；メモリ・ブロックへのポインタ（メモリ管理テーブル＋$10）

返り値

D0.L　　負の数の場合はエラーコード

機能

MEMPTRで指定したメモリ・ブロックを開放します．MEMPTRは$FF48 malloc，$FF7D malloc2で得られたポインタを指定します．

スレッド0以外で管理メモリ領域全体のためのメモリ管理ポインタ＋$10を指定した場合，管理メモリ領域の中のメモリ・ブロックをすべて開放し，現在のスレッドを停止状態にします($FFF9 killと同様な動作)．スレッドのための管理メモリ領域として使用されているメモリ・ブロックが開放されるわけではないことに注意してください．

それ以外の場合は$FF49 mfreeと同様な動作をします．

コール

```
move.l  #MEMPTR,-(SP)
dc.w    $FF7E
addq.l  #4,SP
```

参照

$FF48	malloc(LEN)	
$FF49	mfree(MEMPTR)	
$FF4A	setblock(MEMPTR,LEN)	
$FF7D	malloc2(LEN)	2.0x
$FF7F	setmarea(THREAD,MEMPTR,ALLLEN,LEN)	2.0x
$FFF9	kill()	2.0x

サンプル・プログラム

P.255　SMPL40

$FF7F setmarea(THREAD,MEMPTR,ALLLEN,LEN)仮 2.0x

引数

word　THREAD　；スレッド番号
long　MEMPTR　；メモリ・ブロックへのポインタ（管理テーブル＋$10）
long　ALLLEN　；管理メモリ領域全体のバイト数
long　LEN　；設定する管理メモリ領域の中から確保したいバイト数

返り値

D0.L　設定された管理メモリ領域の中の最初のメモリ管理ポインタ

機能

THREADで指定したスレッドのための管理メモリ領域を設定します．

MEMPTRで指定するのは，$FF48 mallocなどで得られたメモリ管理テーブル＋$10のアドレスです．

ALLLENは管理メモリ領域全体の大きさで，あらかじめ確保してある領域の大きさを超えてはいけません．

LENは設定する管理メモリ領域の中から確保したい領域の大きさで，ALLLENの大きさを超えてはいけません．

このコール以降，THREADで指定されたスレッド中でコールされる$FF7D malloc2，$FF7E mfree2は，設定した管理メモリ領域内を対象に動作することになります．

コール

```
        move.l  #LEN,-(SP)
        move.l  #ALLLEN,-(SP)
        move.l  #MEMPTR,-(SP)
        move.w  #THREAD,-(SP)
        dc.w    $FF7F
        lea     14(SP),SP
```

参照

$FF48　malloc(LEN)
$FF49　mfree(MEMPTR)
$FF4A　setblock(MEMPTR,NEWLEN)
$FF7D　malloc2(MD,LEN)　2.0x
$FF7E　mfree2(MEMPTR)　2.0x

サンプル・プログラム

P.472　ABD.X

$FFF5 getindos( ) 仮 2.0x

引数

なし

返り値

D0.L　　ファンクションコールのInDOSフラグのアドレス

機能

D0にファンクションコールのネスティングレベルを格納している1ワードのワーク(=InDOSフラグ) のアドレスが返ります.

InDOSフラグは，ファンクションコールが呼び出されるときに+1され，ファンクションコール処理から呼び出したプログラムに戻るときに−1されるワークです.

このコールはベクタを書き換えて処理を変更できません.

コール

```
	dc.w	$FFF5
```

サンプル・プログラム

P.261　SPML42

$FFF6 farcall(ADR)仮 2.0x

引数

long　ADR　　；コールするアドレス

返り値

不定

機能

ADRで指定するアドレスをサブルーチンコールします.

コール先へはスーパーバイザモードで制御が移ります（戻ってきたときには元の状態に戻ります).

このコールはベクタを書き換えて処理を変更できません.

コール

```
	move.l	#ADR,-(SP)
	dc.w	$FFF6
	addq.l	#4,SP
```

サンプル・プログラム

P.263　SPML43

$FFF7 memcpy[SOURCE,DISTI,MODE]仮　2.0x

引数

long　SOURCE　；転送元のアドレス
long　DISTI　；転送先のアドレス
word　MODE　；このコールの動作を指定します

返り値

D0.L　＝0　正常終了
　＝1　DISTIアクセス時にバスエラーが発生した
　＝2　SOURCEアクセス時にバスエラーが発生した
　＝－1　おかしなモードを指定したか，奇数アドレスをアクセスしようとした(MODE＝2,4)時0

機能

MODEで指定したサイズのデータを転送します．
バスエラーやアドレスエラーが発生してもエラーウィンドウはオープンされず，返り値で通知されます．
SOURCE，DISTIにはスーパーバイザ領域も指定できます．

★ MODEが1の場合
SOURCEで示されるアドレスに格納されている1バイトのデータをDISTIへ転送します．

★ MODEが2の場合
SOURCEで示されるアドレスに格納されている1ワードのデータをDISTIへ転送します．

★ MODEが4の場合
SOURCEで示されるアドレスに格納されている1ロングワードのデータをDISTIへ転送します．

このコールはベクタを書き換えて処理を変更できません．

コール

```
move.w  #MODE,-(SP)
move.l  #DISTI,-(SP)
move.l  #SOURCE,-(SP)
dc.w    $FFF7
lea     10(SP),SP
```

サンプル・プログラム

P.264　SMPL44

$FFF8 newthread[NAME,LVL,BUSP,BSSP,BSR, ENTRY,BUFF,SLEEP] 仮　2.0x

引数

```
long    NAME   ;スレッド名（16バイト以内）を指すポインタ
word    LVL    ;スレッドのレベル（1～255)
long    BUSP   ;起動するスレッドのためのUSP
long    BSSP   ;起動するスレッドのためのSSP
word    BSR    ;起動するスレッドのためのSR
long    ENTRY  ;スレッドのエントリ・アドレス
long    BUFF   ;プロセス間通信バッファを指すポインタ
long    SLEEP  ;初期スリープ値（0～$FFFFFFFF．0の場合は無期限にスリープする）
```

返り値

D0.l　スレッド番号．負の数の場合はエラーコード

機能

新しいスレッドを起動します．

BUFFで示すバッファは次のような初期構造をしている必要があります．

```
BUFF:   dc.l    LEN     * バッファのバイト数
        dc.l    BODY    * バッファ本体があるアドレス
        dc.w    $0000   * メッセージアトリビュート
        dc.w    $FFFF   * メッセージ発信者ID
BODY:    (実際にはBODYはBUFFの直後でなくてもよい)
        ds.b    LEN
```

SLEEPとして0を指定した場合，スレッドは最初無期限のスリープ状態になります．他のスレッドから$FFFD messageを使ってスリープ解除しないかぎり，動き出すことはありません．

このコールはベクタを書き換えて処理を変更できませんが，新しいスレッドを起動した直後，$FFF8のベクタにしたがってコールを行ないます．
（デフォルトではなにもせずリターンする）．

コール

```
        move.l  #SLEEP,-(SP)
        pea     BUFF
        pea     ENTRY
        move.w  SR,-(SP)
        pea     BSSP_TOP
        pea     BUSP_TOP
        move.w  #LVL,-(SP)
        pea     NAME
        dc.w    $FFF8
        lea     $1C(SP),SP
```

```
                :
NAME:
        dc.b    'BACK GROUND1',0
BUFF:
        dc.l    LEN
        dc.l    BODY
        dc.w    0
        dc.w    $FFFF
BODY:
        dc.b    LEN
                :
SSSP_TOP:
        ds.b    2048    ; システム・スタック領域
USSP_TOP:
        ds.b    2048    ; ユーザースタック領域
```

参照

$FFF9	kill()	2.0x
$FFFA	getthread(THREAD, BUF)	2.0x
$FFFB	suspend(THREAD)	2.0x
$FFFC	sleep(TIME)	2.0x
$FFFD	message(SENDER, RECEIVER, ATR, MESPTR, LEN)	2.0x
$FFFE	getsystimer()	2.0x
$FFFF	juggle()	2.0x

サンプル・プログラム

P.472　ABD.X

$FFF9 kill[ ] 仮 2.0x

引数

なし

返り値

なし

機能

現在のスレッドを停止状態にし，バックグラウンド・タスク・マネージャに戻ります．以降，このスレッドが呼び出されることはありません．また，このスレッドによって動作していたプロセスは，メモリから解放されます．

主スレッド (Human68k System) は，このコールを呼び出してはいけません．(呼び出してもよいが，他のスレッドが生成されていない場合，ハングアップすることになる)．

このコールはベクタを書き換えて処理を変更できませんが，スレッドを停止状態にした直後，$FFF9のベクタにしたがってコールを行ないます．(デフォルトではなにもせずリターンする)．

コール

```
        dc.w    $FFF9
```

参照

$FFF8	newthread(NAME, LVL, BUSP, BSSP, BSR, ENTRY, BUFF, SLEEP)	2.0x
$FFFA	getthread(THREAD, BUF)	2.0x
$FFFB	suspend(THREAD)	2.0x
$FFFC	sleep(TIME)	2.0x
$FFFD	message(SENDER, RECEIVER, ATR, MESPTR, LEN)	2.0x
$FFFE	getsystimer()	2.0x
$FFFF	juggle()	2.0x

サンプル・プログラム

P.472 ABD.X

$FFFA getthread(THREAD,BUF) 仮 2.0x

引数

word　THREAD　；スレッド番号
long　BUF　；スレッドの情報が返るバッファへのポインタ

返り値

D0.L　スレッド番号．負の数の場合はエラーコード
(BUF).b～　スレッドの情報

機能

THREADで指定したスレッドの情報を，BUFで指定したバッファにコピーします．元々のスレッド情報は$7Cバイトありますが，コピーされてくるのはそのうちの先頭$74バイトです．したがって，バッファは$74バイト用意する必要があります．

THREADは原則として0～31の値をとりますが，－1，－2を指定することで特殊な機能を利用できます．

★ THREADが－1の場合

BUFで指定するバッファの中に，スレッド名だけを記入した仮のスレッド情報バッファを作成しておいてこのファンクションをコールすることにより，合致する名前を持つスレッドの情報がBUFで指定するバッファの上に重ねてコピーされてきます．

同時に，そのプロセスのIDナンバーもD0に返ります．指定した名前のスレッドが存在しない場合，－1が返ります．

★ THREADが－2の場合

BUFで指定するバッファの中に，現在のスレッド(自分)の情報をコピーします．同時に，自分のスレッド番号がD0に返ります．

このコールはベクタを書き換えて処理を変更できません．

コール

```
        pea     BUF
        move.w  #THREAD,-(SP)
        dc.w    $FFFA
        addq.l  #6,SP
                :
BUF:
        ds.b    $74
```

参照

$FFF8	newthread(NAME, LVL, BUSP, BSSP, BSR, ENTRY, BUFF, SLEEP)	2.0x
$FFF9	kill()	2.0x
$FFFB	suspend(THREAD)	2.0x
$FFFC	sleep(TIME)	2.0x
$FFFD	message(SENDER, RECEIVER, ATR, MESPTR, LEN)	2.0x
$FFFE	getsystimer()	2.0x
$FFFF	juggle()	2.0x

サンプル・プログラム

P.472　ABD.X

$FFFB suspend(THREAD) 仮　　2.0x

引数

word　THREAD　;スレッド番号

返り値

D0.l　負の数の場合はエラーコード

機能

THREADで指定したスレッドをサスペンド状態にします.

サスペンド状態は無期限のスリープ状態とほぼ同様ですが,他のスレッドからの$FFFD messageによってサスペンドを解除された場合も,それまでのレジスタの値に変化はありません.

このコールはベクタを書き換えて処理を変更できません.

コール

```
move.w  #THREAD,-(SP)
dc.w    $FFFB
addq.l  #2,SP
```

参照

$FFF8	newthread(NAME,LVL,BUSP,BSSP,BSR,ENTRY,BUFF,SLEEP)	2.0x
$FFF9	kill()	2.0x
$FFFA	getthread(THREAD,BUF)	2.0x
$FFFC	sleep(TIME)	2.0x
$FFFD	message(SENDER,RECEIVER,ATR,MESPTR,LEN)	2.0x
$FFFE	getsystimer()	2.0x
$FFFF	juggle()	2.0x

サンプル・プログラム

P.472　ABD.X

$FFFC sleep[TIME] 仮　　2.0x

引数

long　TIME　；スリープカウンタにセットする値

返り値

D0.L　スリープカウンタの残り

機能

スリープカウンタにカウンタ値TIMEを設定し，スリープ状態に入ります．TIMEとして0を指定した場合，無期限のスリープ状態となります．

このコールから戻ってくる場合には以下の2つがあります．

● スリープを正常に終了した場合

スリープカウンタに指定しただけの時間が経過し，スリープ状態から復帰した場合．この場合，D0には0が返ります．

● 他のスレッドから強制的にスリープ解除された場合

他のスレッドの$FFFD messageによって強制的にスリープを解除された場合．この場合，D0にはスリープカウンタの残りが入ります（無期限のスリープだった場合は0）．

スリープカウンタはバックグラウンド・モニタがスレッドを切り替える度に－1され，0になった時点でスリープを正常に終了します．

このコールはベクタを書き換えて処理を変更できません．

コール

```
move.l  #TIME,-(SP)
dc.w    $FFFC
addq.l  #4,SP
```

参照

$FFF8	newthread(NAME, LVL, BUSP, BSSP, BSR, ENTRY, BUFF, SLEEP)	2.0x
$FFF9	kill()	2.0x
$FFFA	getthread(THREAD, BUF)	2.0x
$FFFB	suspend(THREAD)	2.0x
$FFFD	message(SENDER, RECEIVER, ATR, MESPTR, LEN)	2.0x
$FFFE	getsystimer()	2.0x
$FFFF	juggle()	2.0x

サンプル・プログラム

P.472　ABD.X

$FFFD message(SENDER,RECEIVER,ATR, MESPTR,LEN) 仮 2.0x

引数

word　SENDER　；発信側のスレッド番号
word　RECEIVER；受信側のスレッド番号
word　ATR　　；メッセージの付加情報
long　MESPTR　；送信するメッセージの格納されているバッファへのポインタ
long　LEN　　；送信メッセージのバイト数

返り値

D0.L　負の数の場合はエラーコード

機能

RECIEVERで指定したスレッドにMESPTRで示すバッファに格納されたメッセージをLENバイト送信します．

受信側に用意されているメッセージバッファ容量よりも長いメッセージ送ってはいけません．

受信側のスレッドがスリープ/サスペンド状態であった場合，スリープ/サスペンドは解除されます．

ATRはメッセージとともに受信側のスレッドに送られる値です．その値は任意ですが，特定の値を指定した場合，特殊な動作を行ないます．

★ATRが$FFFBの場合

メッセージは送信せず，RECEIVERで指定するスレッドのスリープ／サスペンドの解除のみを行ないます．指定したスレッドが存在しない場合はエラー－1が返ります．この場合，MESPTR，LENの値に意味はありません．

このコールはベクタを書き換えて処理を変更できません．

コール

```
        move.l  #LEN,-(SP)
        pea     MESPTR
        move.w  #ATR,-(SP)
        move.w  #RECEIVER,-(SP)
        move.w  #SENDER,-(SP)
        dc.w    $FFFD
        lea     14(SP),SP
                :
MESPTR:
        dc.b    ....
```

参照

$FF55	common(MD, NAME, ???)	2.0x
$FFF8	newthread(NAME, LVL, BUSP, BSSP, BSR, ENTRY, BUFF, SLEEP)	2.0x
$FFF9	kill()	2.0x
$FFFA	getthread(THREAD, BUF)	2.0x
$FFFB	suspend(THREAD)	2.0x
$FFFC	sleep(TIME)	2.0x

$FFFE	getsystimer()	2.0x
$FFFF	juggle()	2.0x

サンプル・プログラム

P.472　ABD.X

$FFFE getsystimer[ ] 仮　　2.0x

引数

なし

返り値

D0.l　　システム・タイマーの値

機能

タイムスライスごとに+1されるロングワードのカウンタの値を得ます.

CONFIG.SYSの"PROCESS="の第3パラメータで10を指定した場合, このカウンタが以前調べたときから+5されていたならば, 約50msが経過したことになります.

このコールはベクタを書き換えて処理を変更できません.

コール

```
        dc.w    $FFFE
```

参照

$FFF8	newthread(NAME, LVL, BUSP, BSSP, BSR, ENTRY, BUFF, SLEEP)	2.0x
$FFF9	kill()	2.0x
$FFFA	getthread(THREAD, BUF)	2.0x
$FFFB	suspend(THREAD)	2.0x
$FFFC	sleep(TIME)	2.0x
$FFFD	message(SENDER, RECEIVER, ATR, MESPTR, LEN)	2.0x
$FFFF	juggle()	2.0x

サンプル・プログラム

P.472　ABD.X

$FFFF juggle[ ] 仮 2.0x

引数

なし

返り値

なし

機能

現在のスレッド(自分)を中断し，バックグラウンド・モニタに制御を戻します．

このコールを使わなくても，スレッドに与えられたタイムスライスを使い終われば自動的にバックグラウンド・モニタに戻り，次のスレッドの処理の続きを行なうことになります．しかし，次のような場合はこのコールを利用した方がシステム全体のスループットが上がります．

- キーボードなどのデバイスの入力待ちの場合
  入力待ちのループの中にこのコールをはさむことで，遅いデバイスからの応答時間を他のタスクのために使うことができます．
- 他のスレッドと同期をとる場合
  他のスレッドにメッセージを送れる状態になるまで待つ場合や，他のスレッドからのメッセージを待つ場合など．待ちループの中にこのコールを挿入します．
- ○次にMPUの使用権がまわってくるまですることがない場合

このコールはベクタを書き換えて処理を変更できませんが，次のスレッドに切り替えた直後，$FFFFのベクタにしたがってコールを行ないます．
(デフォルトではなにもせずリターンする)

コール

```
        dc.w    $FFFF
```

参照

$FFF8	newthread(NAME, MD, BUSP, BSSP, BSR, ENTRY, BUFF, SLEEP)	2.0x
$FFF9	kill()	2.0x
$FFFA	getthread(THREAD, BUF)	2.0x
$FFFB	suspend(THREAD)	2.0x
$FFFC	sleep(TIME)	2.0x
$FFFD	message(SENDER, RECEIVER, ATR, MESPTR, LEN)	2.0x
$FFFE	getsystimer()	2.0x

サンプル・プログラム

P.472 ABD.X

メッセージの付加情報の意味

$FFFD messageで他のスレッドに送るメッセージは，決められた長さ以内のメッセージ本体と1ワードの付加情報で構成されることは説明しました．

この付加情報の意味ですが，完全にユーザーに任されているというわけではないようです．次の2つの事実に注目してください．

(1)付加情報として$FFFDを指定した場合，メッセージは送信されず，送り先のスレッドのスリープ/サスペンドが解除される．
　→$FFFDはファンクションコールsuspendの番号．
(2)TIMER.Xにおいて，オプションとして"/kill"を指定した場合，TIMER.Xの常駐部を動作させているスレッドには$FFF9の付加情報付のメッセージが送られる．
　→$FFF9はファンクションコールkillの番号．

このことから，どうやら付加情報として$FFxxという値を指定した場合，付加情報と同じ番号のファンクション・コールに関係した動作を指示することになるようです．

付加情報に$FFFDを指定した以外はシステムは関知しないので，ユーザーが対応しなければなりません．

3章 日本語入力フロント・プロセッサ

Humanにはすでにいくつかの日本語フロント・プロセッサが登場していますが，ここでは標準で付属するASK68KVer1.0x/Ver2.0x（以下それぞれASK1.0x/ASK2.0x）についてのデータを掲載することにします．

これらのフロント・プロセッサはいずれもデバイス・ドライバの形でHumanに組み込まれ，新しいCONデバイスとして機能します．組み込みときには，ファンクションコールの$FF22 knjctrl，$FF24 keyctrlのベクタを書き換え，フロント・プロセッサ用のルーチンに置き換えるという処理も行なうことになっています．

$FF22 knjctrlで呼び出される機能は，特にFPコールと呼ばれます．

3.1 フロント・プロセッサのユーザーインターフェイス

フロント・プロセッサのユーザーインターフェイスはHumanによってある程度規定されており，フロント・プロセッサは原則として変換中の表示，候補表示，モード表示などに$FF18 hendspを使うことになっています．そのため，普通にフロント・プロセッサを使っている場合は，かならず画面最下行がフロント・プロセッサ用として使用されます．

これでは困る場合の対応について，プログラマーズ・マニュアルの記述を補足しておきます．

①$FF18 hendspをアプリケーションでエミュレートする

フロント・プロセッサの表示関係をすべて受け持っている$FF18 hendspのベクタを書き換え，アプリケーションの中の別の表示ルーチンと置き換えることで，表示を変更する方法です．

この場合，$FF18 hendspに替わる表示ルーチンさえ用意しておけば，その他の入出力関係はフロント・プロセッサがすべて処理してくれます．

②$FF22 knjctrlでFPの漢字変換カーネルを呼び出す

Humanのキー入力関係のファンクションでは，CTRL＋XF1（ASK2.0xの場合は変更可能）を押した場合，$FF18 hendspを利用する状態でフロント・プロセッサが呼び出されてしまいます．これを避けるために，あらかじめアプリケーションでFPコールの7番を利用して，キーボードからの指示ではフロント・プロセッサを呼び出せないようにしておき，ユーザーのキー入力に対し，アプリケーションが表示や変換作業（FPコールで漢字変換カーネルを利用）を行なうことで，表示を変更します．

この場合，キーの入力や変換状況の表示なども，すべてアプリケーション側で受け持つことになります．次の項でもう少し詳しく解説したいと思います．

3.2 フロント・プロセッサの漢字変換カーネル

フロント・プロセッサのかな漢字変換モジュールのカーネル部はFPコールを通してユーザーに開放されています．FPコールの19～27，33～40がそれで，これらを利用することでユーザーは独自のユーザーインターフェイスでかな漢字変換を行なうことが可能になります．

これらを利用して，「一括変換，辞書先読みなし」，「一括変換，辞書先読みあり」，そして「逐次変換」の3つの方式でかな漢字変換を行なう手順を解説します．

★前準備

FPコール1,7を使って，キー入力を通常入力状態にロックしておきます．

☆一括変換，辞書先読みなし

最も基本的な変換方法です．変換作業を変換開始時にまとめて行なうため，若干待たされる場合があります．

変換ウィンドウのモード表示部に「一括」と表示されている場合は，この辞書先読みなしの一括変換ではなく，辞書先読みありの一括変換を行なっています．

①キー入力

キーボードからの入力を受け付けます．この際のエコーバックなどはHumanに行なわせてもいいのですが，かな漢字変換前の読みの入力であるわけですから，色を変えて表示させるなり，専用のウインドウの中に表示させるなりした方が望ましいでしょう．

ユーザーが変換を指示するまで，入力された読みはバッファに納まっています．ただし，変換前/変換後の文字列は79バイト以下の制限があります．

②変換開始

ユーザーから変換開始の指示があった場合，キー入力結果を納めたバッファを指定してFPコール19で変換を開始します．コールの結果，最初の文節の第１番目の候補と候補の数，そして後続文節を得ることができます．アプリケーションはこれらの表示を行ないます．

フロント・プロセッサ内部にはこの時点でいくつかの候補群（候補ブロック）が用意されています．

③候補選択

ユーザーから候補選択指示があった場合，FPコール20, 21を使って，②で得られた最初の文節の前候補/次候補を選択します．これらも表示を行なってユーザーに選択をうながす必要があるでしょう．

それでもまだ目的の候補が得られない場合は，FPコール22, 23を使って前候補/次候補ブロックを得てから同様にして前候補/次候補を選択します．

④文節移動

ユーザーから別の文節への移動の指示があった場合，FPコール35, 36を使って前文節/次文節に移動します．コールの結果，移動先の文節の第１番目の候補と後続文節を得ることができます．アプリケーションはこれらの文字列を利用して文節の移動を表現する必要があります．

文節移動後はFPコール20, 21, 22, 23を使った場合，移動先の文節に対して前候補/次候補/前候補ブロック/次候補ブロックを得ることになります．

⑤文節伸張

ユーザーから文節伸張の指示があった場合，FPコール33, 34を使って現在対象となっている文節を伸張，再変換します．アプリケーションは文節伸張，再変換した結果を表示します．

⑥全体確定

ユーザーから全体確定の指示があった場合，FPコール24を使ってすべての文節を確定します．アプリケーションはそれまでの入力文字列の表示，候補表示などを消去し，コールの結果得られた最終的な確定文字列を表示します．

以上でひととおりの変換作業は終了です．ふたたび①に戻り，次の変換作業を始めます．

☆一括変換，辞書の先読みあり

外見は辞書の先読みなしの一括変換と変わりませんが，内部的にはむしろ逐次変換に近い方法で変換します．

文字列が入力されると同時にあらかじめ辞書を先読みしますから，変換開始時の作業時間が短縮できます．

変換ウィンドウのモード表示部に「一括」と表示されている場合は，この辞書先読みありの一括変換を行なっています．

①キー入力/先読み

キーボードからの入力を受け付けます．

ユーザーが変換を指示するまで，入力された読みはバッファに納めておきます．それと同時に，modeとして１を指定してFPコール26, 27を呼び出し，フロント・プロセッサ内部のバッファに１文字ずつ（複数文字ずつでも可）文字列を追加して行きます(27の場合，削除します)．文字(文字列)が追加（削除）されると同時に，内部では辞書の先読みが行なわれています．

コールの結果，入力されたものと同じ文字（文字列）が返りますので，それを表示することでエコーバックを実現できます．

変換前/変換後の文字列は79バイト以下の制限がありますので注意してください．変換前文字列が内部バッファの中で79文字を超えた場合，FPコー

ル26はエラーコードを返します．この場合，ユーザーに対して警告を発し，それ以上の文字の追加を禁止する必要があります．

②変換開始

ユーザーから変換開始の指示があった場合，bufとして$00000000を指定してFPコール19を呼び出します．コールの結果，最初の文節の第１番目の候補と候補の数，そして後続文節を得ることができます．アプリケーションはこれらの表示を行ないます．

以降は辞書先読みなしの一括変換と同様です．先読みなしの一括変換の③以降を参照してください．

☆逐次変換

入力する端から自動的に変換していく変換方法です．辞書先読みありの一括変換に近い方法で変換を行ないます．

辞書先読みありの一括変換との違いは以下の２点です．

・自動的に行なわれた変換結果を表示する．
・ある一定長以上の読みを入力した場合，先頭から自動的に確定していく．

①キー入力/逐次変換

キーボードからの入力を受け付けます．

ユーザーが変換を指示するまで，入力された読みはバッファに納めておきます．それと同時に，modeとして０を指定してFPコール26, 27を呼び出し，フロント・プロセッサ内部のバッファに１文字ずつ（複数文字ずつでも可）文字列を追加して行きます(27の場合，削除します)．文字(文字列)が追加（削除）されると同時に，内部では自動的に変換が行なわれています．

コールの結果，途中の変換結果が返りますので，それを表示することでエコーバックを実現できます．

変換前/変換後の文字列は79バイト以下の制限があるので注意してください．変換前文字列が内部バッファの中で79文字を超えた場合，FPコール26はエラーコードを返します．この場合，FPコール25で先頭文節から部分確定していく必要があります．

②部分確定

入力バッファがいっぱいになりかけた場合，先頭文節を自動的に部分確定してバッファから追い出す必要があります．

部分確定の手順は次のとおりです．

a)no, buf, kouho, kouzoku, nkouhoなどのパラメータを指定してFPコール25を呼んで部分確定．

b)FPコール25は，確定した文節＋次文節をbufで指定したバッファに返します．kouhoで指定したバッファの中に入っている文字数を計算して，確定した文節だけを取り出し，変換が完了した文字列として扱います．これで一応部分確定は終了です．

c)FPコール25は，D0に確定した文節の読みのバイト数を返します．それを使って，アプリケーションが読みの入力を保存しておいたバッファの中の，次文節が読みが始まるアドレスを求めます．まず，bufとして$00000000を指定してFPコール26を呼び出し，逐次先読みを初期化した上で，次文節のアドレスを指定してFPコール26を呼び出し，①に戻り後続文節の逐次変換を続行します．

③変換開始

ユーザーから変換開始の指示があった場合，bufとして$00000000を指定してFPコール19を呼び出します．コールの結果，最初の文節の第１番目の候補と候補の数，そして後続文節を得ることができます．アプリケーションはこれらの表示を行ないます．

以降は辞書先読みなしの一括変換と同様です．先読みなしの一括変換の③以降を参照してください．

3.3 FPコールの呼び出し方

FPコールを呼び出す場合は一般的に次の手順に従います．

①各FPコールに渡すための値をユーザースタックに積む．
②FPコール番号をユーザースタックに積む．
③$FF22 knjctrlをコールする．
④戻ってきたらユーザースタック・ポインタを補正．

FPコールを使う上で右記のような事項に注意してください．

①FPコールに与えるパラメータは，FPコール番号を含めてすべてロングワードです．
②レジスタはD0以外すべて保存されます．
③FPコールの呼び出しも，マクロを作っておくと便利です．これは，その一例です．

```
FPFNC  macro    number
       move. 1  # number,－(SP)
       dc. w    $ff22
       endm
```

（このマクロの場合，スタック・ポインタはFPコール番号の分も含めて補正する必要があります）

3.4 ASK68KVer1.0x のFPコール使用法解説

Humanの標準フロント・プロセッサであるASK68KVer1.0xのFPコールの解説を番号順に行ないます．ASK1.0xは基準となるべきフロント・プロセッサですから，他のフロント・プロセッサもASK1.0xのFPコールとのコンパチビリティを維持することが望ましいと思われます．

FPコール13 KNJCTRL(13,source,dist)

(FPコール番号)　(FPコール引数リスト)

引数（引数の名前と内容）

source　半角文字列を指すポインタ
dest　変換された全角文字列が返るバッファを指すポインタ

返り値（返り値のはいるレジスタ，バッファとその内容）

D0.L　＝0　正常終了
　<> 0　変換エラーが起きた文字の，バッファ先頭からのバイト数
(dist).b～　全角に変換された文字列

機能（機能の概略）

半角文字列を全角文字列に変換します．
文字列は変換前，変換後共に79バイト以下で，終端に$00が置かれているものとします．

コール（一般的なコール方法）

```
        pea     dist
        pea     source
        move.l  #13,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
source:
        dc.b    'ABCDEF',0
dist:
        ds.b    80
```

FPコール1　KNJCTRL(1,mode)

引数

mode　かな漢字変換モード
　=0　通常入力モード
　=1　一括変換，辞書の先読みなし
　=2　一括変換，辞書の先読みあり
　=3　逐次変換

返り値

D0.L　=0　正常終了
　=1　ドライブの準備ができていません
　=2　モードの誤り

機能

かな漢字変換モード，通常入力モードを切り替えます．

modeが1, 2, 3の場合は，画面の最下行がかな漢字変換ウインドウとなります．modeが0の場合はかな漢字変換ウインドウがクローズされ，通常入力モードに戻ります．

コール

```
move.l  #mode, -(SP)
move.l  #1, -(SP)
dc.w    $FF22
addq.l  #8, SP
```

FPコール2　KNJCTRL(2)

引数

なし

返り値

D0.L　かな漢字変換モード
　=0　通常入力モード
　=1　一括変換，辞書の先読みなし
　=2　一括変換，辞書の先読みあり
　=3　逐次変換

機能

かな漢字変換モードに入った場合のかな漢字変換モードを返します．

このコールを呼び出した時点で通常入力モードであった場合も，返り値として0は返らず，かな漢字変換モードに入った場合のモードが返ります．

コール

```
move.l  #2, -(SP)
dc.w    $FF22
addq.l  #4, SP
```

FPコール3　KNJCTRL(3,mode)

引数

mode　フラッシュモード
　=0　フラッシュモードの解除
　=1　フラッシュモードの設定

返り値

D0.L　=0　正常終了
　=2　モード指定の誤り

機能

フラッシュモードを設定/解除します．

フラッシュモードを設定した状態の場合，$FF0C kflushでキー入力を行なう場合，フロント・プロセッサのバッファも空にします．フラッシュモードを解除した状態の場合は，フロント・プロセッサのバッファは空になりません．

コール

```
move.l  #mode, -(SP)
move.l  #3, -(SP)
dc.w    $FF22
addq.l  #8, SP
```

FPコール4　KNJCTRL(4)

引数

なし

返り値

D0.L　現在のフラッシュモード
　=0　解除されている
　=1　設定されている

機能

現在のフラッシュモードを返します.

コール

```
move.l  #4,-(SP)
dc.w    $FF22
addq.l  #4,SP
```

FPコール5　KNJCTRL(5,mode)

引数

mode　入力モード

bit$F		bit$3	bit$2	bit$1	bit$0
0		0			

bit$2　0:ローマ字変換/1:通常モード
bit$1　0:ひらがな/1:カタカナ
bit$0　0:全角/1:半角

返り値

D0.L　=0　正常終了
　=2　モード指定の誤り

機能

modeのビット・パターンによってフロント・プロセッサの入力モードを設定します.

プログラマーズ・マニュアルのビット配置は一部間違っています. 注意してください.

コール

```
move.l  #mode,-(SP)
move.l  #5,-(SP)
dc.w    $FF22
addq.l  #8,SP
```

FPコール6　KNJCTRL(6)

引数

なし

返り値

D0.L　入力モード

機能

フロント・プロセッサの現在の入力モードを返します.

返り値の意味は, FPコール5番の引数と同様です. bit$3, $2, $1以外のビットには意味がありませんので無視して結構です.

コール

```
move.l  #6,-(SP)
dc.w    $FF22
addq.l  #4,SP
```

FPコール7　KNJCTRL(7,mode)

引数

mode　かな漢字変換モードのロック
　=0　かな漢字変換モードをロック
　=1　かな漢字変換モードのロック解除

返り値

D0.L　=0　正常終了
　=2　モード指定の誤り

機能

キーボードからの指示によってかな漢字変換モードから出入りすることを許可するか，禁止するかの設定を行ないます．

ロックした場合，キーボードからの指示は無視されるようになります．ロックを解除した場合，キーボードからの指示でかな漢字変換モードから出入りすることが許可されます．

かな漢字変換カーネルを呼び出す場合，かな漢字変換モードをロックしておく必要があります．

コール

```
move.l  #mode,-(SP)
move.l  #7,-(SP)
dc.w    $FF22
addq.l  #8,-(SP)
```

FPコール8　KNJCTRL(8)

引数

なし

返り値

D0.L　現在のかな漢字変換モードロック状態
　=0　ロックされている
　=1　ロック解除されている

機能

現在のかな漢字変換モードのロック状態を返します．

コール

```
move.l  #8,-(SP)
dc.w    $FF22
addq.l  #4,SP
```

FPコール9　KNJCTRL(9,mode)

引数

mode　JIS/句点モードの指定
　=0　JIS/シフトJISモード
　=1　句点モード

返り値

D0.L　=0　正常終了
　=2　モード指定の誤り

機能

コード入力時の漢字コード体系を指定します．

コール

```
move.l  #mode,-(SP)
move.l  #9,-(SP)
dc.w    $FF22
addq.l  #8,SP
```

FPコール10　KNJCTRL(10)

引数

なし

返り値

D0.L　コード入力時のモード
　=0　JIS/シフトJISモード
　=1　句点モード

機能

現在設定されているコード入力時の漢字コード体系を返します．

コール

```
move.l  #10,-(SP)
dc.w    $FF22
addq.l  #4,SP
```

FPコール11　KNJCTRL(11,mode)

引数

mode　学習モード
　=0　メモリに学習
　=1　ディスクに学習

返り値

D0.L　=0　正常終了
　=1　モード指定の誤り

機能

フロント・プロセッサの学習モードを指定します.

コール

```
move.l  #mode,-(SP)
move.l  #11,-(SP)
dc.w    $FF22
addq.l  #8,SP
```

FPコール12　KNJCTRL(12)

引数

なし

返り値

D0.L　現在の学習モード
　=0　メモリに学習
　=1　ディスクに学習

機能

現在の学習モードを返します.

コール

```
move.l  #12,-(SP)
dc.w    $FF22
addq.l  #4,SP
```

FPコール13　KNJCTRL(13,source,dist)

引数

source　半角文字列を指すポインタ
dist　変換された全角文字列が返るバッファを指すポインタ

返り値

D0.L　=0　正常終了
　<> 0　変換エラーが起きた文字の,バッファ先頭からのバイト数
(dist).b〜　全角に変換された文字列

機能

半角文字列を全角文字列に変換します.

文字列は変換前,変換後共に79バイト以下で,終端に$00が置かれているものとします.

コール

```
        pea     dist
        pea     source
        move.l  #13,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
source:
        dc.b    'ABCDEF',0
dist:
        ds.b    80
```

FPコール14 KNJCTRL(14,source,dist)

引数

source　全角文字列を指すポインタ

dist　変換された半角文字列が返るバッファを指すポインタ

返り値

D0.L　=0　正常終了

<> 0　変換エラーが起きた文字の，バッファ先頭からのバイト数

(dist).b～　半角に変換された文字列

機能

全角文字列を半角文字列に変換します．

文字列は変換前，変換後共に79バイト以下で，終端に$00が置かれているものとします．

コール

```
        pea     dist
        pea     source
        move.l  #14,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
source:
        dc.b    'ABCDEF',0
dist:
        ds.b    80
```

FPコール15 KNJCTRL(15,source,dist)

引数

source　半角文字列を指すポインタ

dist　ローマ字変換された全角カタカナ文字列が返るバッファを指すポインタ

返り値

D0.L　=0　正常終了

<> 0　変換エラーが起きた文字の，バッファ先頭からのバイト数

(dist).b～　全角カタカナに変換された文字列

機能

半角文字列を全角のカタカナ文字列にローマ字変換します．

文字列は変換前，変換後共に79バイト以下で，終端に$00が置かれているものとします．

コール

```
        pea     dist
        pea     source
        move.l  #15,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
source:
        dc.b    'KAKIKUKEKO',0
dist:
        ds.b    80
```

FPコール16　KNJCTRL(16,source,dist)

引数

source　半角文字列を指すポインタ

dist　ローマ字変換された全角ひらがな文字列が返るバッファを指すポインタ

返り値

D0.L　＝0　正常終了

<> 0　変換エラーが起きた文字の，バッファ先頭からのバイト数

(dist).b〜　全角ひらがなに変換された文字列

機能

半角文字列を全角のひらがな文字列にローマ字変換します．

文字列は変換前，変換後共に79バイト以下で，終端に$00が置かれているものとします．

コール

```
        pea     dist
        pea     source
        move.l  #16,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
source:
        dc.b    'SASHISUSESO',0
dist:
        ds.b    80
```

FPコール17　KNJCTRL(17,source,dist)

引数

source　全角カタカナ文字列を指すポインタ

dist　変換された全角ひらがな文字列が返るバッファを指すポインタ

返り値

D0.L　＝0　正常終了

<> 0　変換エラーが起きた文字の，バッファ先頭からのバイト数

(dist).b〜　全角ひらがなに変換された文字列

機能

全角カタカナ文字列を全角ひらがな文字列に変換します．

文字列は変換前，変換後共に79バイト以下で，終端に$00が置かれているものとします．

コール

```
        pea     dist
        pea     source
        move.l  #17,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
source:
        dc.b    'カタカナ',0
dist:
        ds.b    80
```

FPコール18　KNJCTRL(18,source,dist)

引数

source　全角ひらがな文字列を指すポインタ
dist　変換された全角カタカナ文字列が返るバッファを指すポインタ

返り値

D0.L　＝0　正常終了
　　<> 0　変換エラーが起きた文字の，バッファ先頭からのバイト数
(dist).b～　全角カタカナに変換された文字列

機能

全角ひらがな文字列を全角カタカナ文字列に変換します．

文字列は変換前，変換後共に79バイト以下で，終端に$00が置かれているものとします．

コール

```
        pea     dist
        pea     source
        move.l  #18,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
source:
        dc.b    "ひらがな",0
dist:
        ds.b    80
```

FPコール19　KNJCTRL(19,buf,kouho,kouzou)

引数

buf　全角ひらがな文字列を指すポインタ
kouho　最初の候補の入るバッファを指すポインタ
kouzou　後続文節の入るバッファを指すポインタ

返り値

D0.L　候補数．負の数の場合はエラー
(kouho).b～　最初の候補文字列
(kouzou).b～　後続文節の文字列

機能

一括変換，および逐次先読みの，最初の変換候補群と後続文節を作成します．つまり，読みを入力して，最初に変換キーを押した場合の処理を行ないます．

最初の文節は自動的に判断され，その文節に対する第1番目の候補がkouhoで示されるバッファに返ります．ここで得られる候補は，候補ブロック内の候補番号1の候補ということになります．最初の文節以降の文字列も変換されてkouzokuで示されるバッファに返ります．D0には最初の文節に用意された候補の数が返ります．

逐次先読みの場合，bufで指定したで文字列ではなく，FPコール26,27でフロント・プロセッサ内部のバッファに作成された読みバッファの内容が変換の対象となります．この場合，bufとして$0000 0000を指定します．

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください．

変換文字列は79バイト以内です．

コール

```
        pea     kouzoku
        pea     kouho
        pea     buf
        move.l  #19,-(SP)
        dc.w    $FF22
        lea     16(SP),SP
                :
buf:
        dc.b    'へんかんもじれつ',0
kouho:
        ds.b    80
kouzoku:
        ds.b    80
```

FPコール20 KNJCTRL(20,kouho,kouzoku)

引数

kouho　　前候補が格納されるバッファを指すポインタ

kouzoku　後続文節が格納されるバッファを指すポインタ

返り値

D0.L　　=0　候補なし

　　　　<> 0　候補番号

(kouho).b~　前候補

(kouzoku).b~　後続文節

機能

候補群ブロックの中の前候補を得ます.

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください.

コール

```
        pea     kouzoku
        pea     kouho
        move.l  #20,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
kouho:
        ds.b    80
kouzoku:
        ds.b    80
```

FPコール21 KNJCTRL(21,kouho,kouzoku)

引数

kouho　　次候補が格納されるバッファを指すポインタ

kouzoku　後続文節が格納されるバッファを指すポインタ

返り値

D0.L　　=0　候補なし

　　　　<> 0　候補番号

(kouho).b~　次候補

(kouzoku).b~　後続文節

機能

候補群ブロックの中の次候補を得ます.

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください.

コール

```
        pea     kouzoku
        pea     kouho
        move.l  #21,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
kouho:
        ds.b    80
kouzoku:
        ds.b    80
```

FPコール22　KNJCTRL(22,kouho,kouzoku)

引数

kouho　前候補ブロックの最初の候補が格納されるバッファを指すポインタ

kouzoku　後続文節が格納されるバッファ

返り値

D0.L　=0　候補ブロックなし

<> 0　前候補ブロックの候補数

(kouho).b～　前候補ブロックの最初の候補

(kouzoku).b～　後続文節

機能

前候補ブロックを作成し，作成された前候補ブロックの最初の候補を得ます．

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください．

コール

```
        pea     kouzoku
        pea     kouho
        move.l  #22,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
kouho:
        ds.b    80
kouzoku:
        ds.b    80
```

FPコール23　KNJCTRL(23,kouho,kouzoku)

引数

kouho　次候補ブロックの最初の候補が格納されるバッファを指すポインタ

kouzoku　後続文節が格納されるバッファ

返り値

D0.L　=0　候補ブロックなし

<> 0　次候補ブロックの候補数

(kouho).b～　次候補ブロックの最初の候補

(kouzoku).b～　後続文節

機能

次候補ブロックを作成し，作成された前候補ブロックの最初の候補を得ます．

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください．

コール

```
        pea     kouzoku
        pea     kouho
        move.l  #23,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
kouho:
        ds.b    80
kouzoku:
        ds.b    80
```

FPコール24　KNJCTRL(24,buf)

引数

buf　確定後の全体の文字列が格納されるバッファを指すポインタ

返り値

D0.L　=0　正常終了

<> 0　エラー

機能

FPコール19で変換を開始した文字列の全体を確定し，全体の文字列をbufで示されるバッファに返します．

学習結果はこの時点で記録されます．

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください．

コール

```
        pea     buf
        move.l  #24,-(SP)
        dc.w    $FF22
        addq.l  #8,SP
                :
buf:
        ds.b    80
```

FPコール25　KNJCTRL(25,no,buf,kouho,kouzoku,nkouho)

引数

no	先頭文節を確定する候補の候補番号
buf	確定された先頭文節＋次文節が格納されるバッファを指すポインタ
kouho	次文節の第１番目の候補が格納されるバッファを指すポインタ
kouzoku	次文節に続く後続文節が格納されるバッファを指すポインタ
nkouho	次文節の候補数が返る，1ワードのバッファを指すポインタ

返り値

D0.L	＝0	候補番号が不正
	<> 0	確定文節の読みのバイト数
(buf).b～		確定された先頭文節＋次文節
(kouho).b～		次文節の第１番目の候補
(kouzoku).b～		次文節に続く後続文節
(nkouho).w		次文節の候補数（ワード・データ）

機能

先頭文節を部分確定します．逐次変換で，入力バッファがいっぱいになった場合に，先頭文節を確定して吐き出すときなどに使います．

確定後，後続文節の逐次変換を続ける場合は，D0に返った値を利用して，アプリケーション側で保存しておいた読みの入力バッファの中の次文節の読みのアドレスを計算し，FPコール27を呼び出します．

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください．

コール

```
        pea     nkouho
        pea     kouzoku
        pea     kouho
        pea     buf
        move.l  #no,-(SP)
        move.l  #25,-(SP)
        dc.w    $FF22
        lea     24(SP),SP
                :
buf:
        ds.b    80
kouho:
        ds.b    80
kouzoku:
        ds.b    80
nkouho:
        ds.w    1
```

FPコール26 KNJCTRL(26,buf,retbuf,mode)

引数

buf　　内部の読みバッファに追加する文字列を指すポインタ

retbuf　途中の変換文字列が格納されるバッファを指すポインタ

mode　　=0　逐次変換

　　　　=1　一括変換，辞書先読みあり

返り値

D0.L　　=0　正常終了

　　　　<> 0　エラー（変換バッファがいっぱいになった場合など）

(retbuf).b～　途中の変換文字列

機能

辞書先読みありの一括変換，逐次変換で使うFPコールです．フロント・プロセッサ内部の読みバッファに読み文字列を追加します．

bufとして$00000000を指定すると，フロント・プロセッサ内部の読みバッファは初期化されます．最初の読みの入力の直前や，部分確定を行なった直後などにはかならずいちど初期化するようにしてください．

retbufで示されるバッファには，変換途中の文字列が返りますが，modeによって返る内容が違います．

★modeが0の場合（逐次変換）

retbufで示されるバッファには，初期化/全体確定以降に内部の読みバッファに追加された文字列を，漢字に変換した中間結果が返ります．

★modeが1の場合（一括変換，辞書先読みあり）

retbufで示されるバッファには，初期化/全体確定以降に内部の読みバッファに追加された文字列がそのまま返ります．

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください．

buf，retbufで指定されるバッファは，いずれも79バイト以下であることに注意してください．

コール

```
        move.l  #mode,-(SP)
        pea     retbuf
        pea     buf
        move.l  #26,-(SP)
        dc.w    $FF22
        lea     16(SP),SP
                :
buf:
        dc.b    'ついかもじれつ',0
retbuf:
        ds.b    80
```

FPコール27 KNJCTRL(27,n,retbuf,mode)

引数

n　　　内部の読みバッファから削除するバイト数

retbuf　途中の変換結果が格納されるバッファを指すポインタ

mode　　=0　逐次変換

　　　　=1　一括変換，辞書先読みあり

返り値

D0.L　　=0　正常終了

　　　　<> 0　エラー

(retbuf).b～　途中の変換文字列

機能

辞書先読みありの一括変換，逐次変換で使うFPコールです．フロント・プロセッサ内部の読みバッファから文字列を削除します．

内部の読みバッファの終端からnで指定したバイト数だけ文字を削除し，その結果で再変換（再辞書先読み）を行ないます．

retbufで示されるバッファには，再変換された変換途中の文字列が返りますが，modeによって返る内容が違います．

★modeが0の場合（逐次変換）

retbufで示されるバッファには，文字の削除処理を施された内部の読みバッファの内容を漢字に変換した中間結果が返ります．

★modeが1の場合（一括変換，辞書先読みあり）

retbufで示されるバッファには，文字の削除処理を施された内部の読みバッファの内容がそのまま返ります．

このFPコールはキーボードからのモード切り変

えがロックされた状態で呼び出してください．

buf，retbufのどちらで指定されるバッファも79バイト以下であることに注意してください．

コール

```
        move.l  #mode,-(SP)
        pea     retbuf
        move.l  #n,-(SP)
        move.l  #27,-(SP)
        dc.w    $FF22
        lea     16(SP),SP
                :
retbuf:
        ds.b    80
```

FPコール28　KNJCTRL(28)

引数

なし

返り値

D0.L	=0	正常終了
	=1	ドライブの準備ができていない
	その他	エラー

機能

辞書ファイルをオープンします．FPコール19～27，30，31，33～40を使う前には，辞書ファイルをオープンしておく必要があります．

コール

```
        move.l  #28,-(SP)
        dc.w    $FF22
        addq.l  #4,SP
```

FPコール29　KNJCTRL(29)

引数

なし

返り値

D0.L	=0	正常終了
	<>0	エラー

機能

辞書ファイルをクローズします．

コール

```
        move.l  #29,-(SP)
        dc.w    $FF22
        addq.l  #4,SP
```

FPコール30　KNJCTRL(30,fd,yomi,tango,syn)

引数

fd　　単語登録を行なう辞書
　　=0　メイン辞書
　　=1　サブ辞書
yomi　登録する単語の読みの文字列（全角ひらがな，全角英数字）を指すポインタ
tango　登録する単語の文字列を指すポインタ
syn　文法コード

返り値

D0.L　=0　正常終了
　　=1　辞書の種類が不適当
　　=2　文法コードが不適当
　　=3　読みが不適当
　　=4　登録するページがない

機能

辞書に単語を登録します．

文法コードはP.494を参照してください．

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください．

コール

```
        move.l  #syn,-(SP)
        pea     tango
        pea     yomi
        move.l  #syn,-(SP)
        move.l  #30,-(SP)
        dc.w    $FF22
        lea     20(SP),SP
                :
yomi:
        dc.b    'とうろく',0
tango:
        dc.b    '登録',0
```

FPコール31　KNJCTRL(31,fd,yomi,tango,syn)

引数

fd　　単語削除を行なう辞書
　　=0　メイン辞書
　　=1　サブ辞書
yomi　削除する単語の読みの文字列（全角ひらがな，全角英数字）を指すポインタ
tango　削除する単語の文字列を指すポインタ
syn　文法コード

返り値

D0.L　=0　正常終了
　　=0−1　削除すべき単語がなかった
　　=1　辞書の種類が不適当
　　=2　文法コードが不適当
　　=3　読みが不適当
　　=4　辞書がなかった

機能

辞書から単語を削除します．

文法コードはP.494を参照してください．

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください．

コール

```
        move.l  #syn,-(SP)
        pea     tango
        pea     yomi
        move.l  #syn,-(SP)
        move.l  #31,-(SP)
        dc.w    $FF22
        lea     20(SP),SP
                :
yomi:
        dc.b    'さくじょ',0
tango:
        dc.b    '削除',0
```

FPコール32　KNJCTRL(32,maindic,subdic)

引数

maindic　メイン辞書名を指すポインタ

subdic　サブ辞書名を指すポインタ

返り値

D0.L　=0　正常終了

　　　=1　ドライブの準備ができていない

機能

メイン辞書，サブ辞書のファイル名を登録します．ファイル名登録後はFPコール28でオープンしてください．

コール

```
        pea     subdic
        pea     maindic
        move.l  #32,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
maindic:
        dc.b    'A:¥ASK¥X68K_M.DIC',0
subdic:
        dc.b    'D:¥X68K_S.DIC',0
```

FPコール33　KNJCTRL(33,kouho,kouzoku)

引数

kouho　最初の候補が格納されるバッファを指すポインタ

kouzoku　後続文節が格納されるバッファを指すポインタ

返り値

D0.L　候補数

機能

現在変換対象になっている文節を1文字分だけ長くして再変換します．

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください．

コール

```
        pea     kouzoku
        pea     kouho
        move.l  #33,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
kouho:
        ds.b    80
kouzoku:
        ds.b    80
```

FPコール34　KNJCTRL(34,kouho,kouzoku)

引数

kouho　最初の候補が格納されるバッファを指すポインタ

kouzoku　後続文節が格納されるバッファを指すポインタ

返り値

D0.L　候補数

機能

現在変換対象になっている文節を1文字分だけ短くして再変換します．

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください．

コール

```
        pea     kouzoku
        pea     kouho
        move.l  #34,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
kouho:
        ds.b    80
kouzoku:
        ds.b    80
```

FPコール35　KNJCTRL(35,kouho,kouzoku)

引数

kouho　前文節の最初の候補が格納されるバッファを指すポインタ

kouzoku　前文節に続く後続文節が格納されるバッファを指すポインタ

返り値

D0.L　＝0　前文節がない

　　　<> 0　候補数

機能

前文節に移動します.

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください.

コール

```
        pea     kouzoku
        pea     kouho
        move.l  #35,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
kouho:
        ds.b    80
kouzoku:
        ds.b    80
```

FPコール36　KNJCTRL(36,kouho,kouzoku)

引数

kouho　次文節の最初の候補が格納されるバッファを指すポインタ

kouzoku　次文節に続く後続文節の格納されるバッファを指すポインタ

返り値

D0.L　＝0　次文節がない

　　　<> 0　候補数

機能

次文節に移動します.

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください.

コール

```
        pea     kouzoku
        pea     kouho
        move.l  #36,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
kouho:
        ds.b    80
kouzoku:
        ds.b    80
```

FPコール37　KNJCTRL(37,kouho,kouzoku)

引数

kouho　候補が格納されるバッファを指すポインタ

kouzoku　後続文節の格納されるバッファを指すポインタ

返り値

D0.L　=0　正常終了
　　<> 0　エラー

機能

現在変換対象になっている文節をカタカナにします．

このコールは，候補を選択するFPコールと同様に使います．呼び出された結果，最初の候補ブロックが作成され，カタカナ候補が第1番目の候補になります．

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください．

コール

```
        pea     kouzoku
        pea     kouho
        move.l  #37,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
kouho:
        ds.b    80
kouzoku:
        ds.b    80
```

FPコール38　KNJCTRL(38,kouho,kouzoku)

引数

kouho　候補が格納されるバッファを指すポインタ

kouzoku　後続文節の格納されるバッファを指すポインタ

返り値

D0.L　=0　正常終了
　　<> 0　エラー

機能

現在変換対象になっている文節をひらがなにします．

このコールは，候補を選択するFPコールと同様に使います．呼び出された結果，最初の候補ブロックが作成され，ひらがな候補が第1番目の候補になります．

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください．

コール

```
        pea     kouzoku
        pea     kouho
        move.l  #38,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
kouho:
        ds.b    80
kouzoku:
        ds.b    80
```

FPコール39 KNJCTRL(39,kouho,kouzoku)

引数

kouho　候補が格納されるバッファを指すポインタ

kouzoku　後続文節の格納されるバッファを指すポインタ

返り値

D0.L　=0　正常終了
　　　<> 0　エラー

機能

現在変換対象になっている文節を半角文字にします.

このコールは, 候補を選択するFPコールと同様に使います. 呼び出された結果, 最初の候補ブロックが作成され, 半角文字候補が第1番目の候補になります.

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください.

コール

```
        pea     kouzoku
        pea     kouho
        move.l  #39,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
kouho:
        ds.b    80
kouzoku:
        ds.b    80
```

FPコール40 KNJCTRL(40,kouho,kouzoku)

引数

kouho　候補が格納されるバッファを指すポインタ

kouzoku　後続文節の格納されるバッファを指すポインタ

返り値

D0.L　=0　正常終了
　　　<> 0　エラー

機能

現在変換対象になっている文節を全角文字にします.

このコールは, 候補を選択するFPコールと同様に使います. 呼び出された結果, 最初の候補ブロックが作成され, 全角文字候補が第1番目の候補になります.

このFPコールはキーボードからのモード切り変えがロックされた状態で呼び出してください.

コール

```
        pea     kouzoku
        pea     kouho
        move.l  #40,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
kouho:
        ds.b    80
kouzoku:
        ds.b    80
```

FPコール41　KNJCTRL[41,maindic,subdic]

引数

maindic　メイン辞書名が格納されるバッファを指すポインタ

subdic　サブ辞書名が格納されるバッファを指すポインタ

返り値

D0.L　　=0　正常終了

　　　　<> 0　エラー

機能

現在使用中のメイン辞書ファイル名，サブ辞書ファイル名を得ます．

コール

```
        pea     subdic
        pea     maindic
        move.l  #32,-(SP)
        dc.w    $FF22
        lea     12(SP),SP
                :
maindic:
        ds.b    40
subdic:
        ds.b    40
```

3.5　ASK68KVer2.0xのFPコール使用法解説

HumanがVer2.0xにバージョンアップするのに伴い，ASKもバージョンアップされ，ASK68KVer2.0xとなり，変換が高速化され，環境のカスタマイズが可能になるなど，使い勝手の向上が図られています．

FPコールもASK1.0xと互換性を保ちながら拡張されています．この章では，拡張されたFPコールについて解説します．

FPコール50　KNJCTRL[50]

引数

なし

返り値

D0.L　　ASK2.00の場合　200(10)　($000000C8)

　　　　ASK2.01の場合　201(10)　($000000C9)

機能

ASK2.0xのバージョンが返ります．

なお，ASK1.0xでこのFPコールを呼び出した場合，存在しないFPコールを呼び出したとしてD0に−1が返ります．

コール

```
        move.l  #50,-(SP)
        dc.w    $FF22
        addq.l  #4,SP
```

FPコール51　KNJCTRL[51]

引数

なし

返り値

D0.L　　メイン辞書ファイルのバージョン

　　　　ASK1.0x用の辞書ファイルの場合

　　　　100(10)

　　　　ASK2.0x用の辞書ファイルの場合

　　　　200(10)

機能

メイン辞書ファイルを調べ，辞書ファイルのバージョンを返します．

ASK2.0x用のメイン辞書ファイルはASK1.0x用の辞書ファイルに変換スピード向上のための情報が追加されています．ASK1.0xとの互換性は維持されているので，ASK2.0xでもASK1.0xの辞書ファイルを使うことができるようになっています．

コール

```
        move.l  #51,-(SP)
        dc.w    $FF22
        addq.l  #4,SP
```

FPコール52 KNJCTRL(52,envfile)

引数

envfile 環境ファイル名が格納されるバッファを指すポインタ

返り値

D0.L =0 正常終了
<> 0 エラー

機能

envfileで指定した環境ファイルを読み込んで，ASK2.0xの日本語変換環境を設定します．

envfileとして$00000000を指定すると，環境を初期化します．

コール

```
        pea     envfile
        move.l  #52,-(SP)
        dc.w    $FF22
        addq.l  #8,SP
                :
envfile:
        dc.b    'A:¥ASK¥ENV1.ASK',0
```

FPコール53 KNJCTRL(53)

引数

なし

返り値

D0.L 入力されたキャラクタコード．1バイトのキャラクタコードがロングワードに符号拡張されている．

機能

キーが押されている場合はキャラクタコードをロングワードに符号拡張したものを返し，キーが押されていない場合は$00000000が返ります．

エコーバック，ブレークチェックは行ないません．

コール

```
        move.l  #53,-(SP)
        dc.w    $FF22
        addq.l  #4,SP
```

FPコール54 KNJCTRL(54,echomode)

引数

echomode=0 変換行にエコー
=1 カーソル行にエコー

返り値

D0.L =0 正常終了
=2 おかしなモードを指定した
<> 0 エラー

機能

変換中のエコーバックを行なう場所を指定します．

カーソル行へのエコーバックは，$FF18 hendspを拡張することによって実現されているわけではなく，ASK2.0xが直接画面表示ファンクションコール/IOCSを呼び出しています．

コール

```
        move.l  #echomode,-(SP)
        move.l  #54,-(SP)
        dc.w    $FF22
        addq.l  #8,SP
```

FPコール55 KNJCTRL(55)

引数

なし

返り値

D0.L =0 変換行にエコー
=1 カーソル行にエコー

機能

現在のエコーモードを返します．

コール

```
        move.l  #55,-(SP)
        dc.w    $FF22
        addq.l  #4,SP
```

FPコール56　KNJCTRL(56,mode)

引数

mode	＝0	エコーモードをロック
	＝1	エコーモードのロック解除

返り値

D0.L	＝0	正常終了
	＝2	おかしなモードを指定した
	<> 0	エラー

機能

キーボードからのエコーモードの切り替えをロックしたり，ロックを解除したりします．

コール

```
move.l  #mode, -(SP)
move.l  #56, -(SP)
dc.w    $ff22
addq.l  #8, SP
```

プログラミングに便利なツール類(II)

● nl5_iocs (作:beeps氏)

X68KのHuman/IOCSを使った文字表示は若干遅く，なにかとせっかちになっているプログラミング時にはイライラのもととなります．

nl5_iocsは，IOCSの(ひいてはHumanの)文字表示ルーチンをチューンアップして超高速にするばかりか，好みのフォントの定義，縮小表示，エスケープシーケンスの拡張など，多彩な機能を付加してくれます．

このソフトはフリーウェアとしてPEKINなどで入手できます．

さらに強力なn18_iocsも登場しました．

●セミオート・ディスアセンブラ・ds (作:たこ太郎氏)

他人のプログラムを読んで参考にするのは，プログラミングの力をつけるためには非常に有効な手段です．ソースが付属している場合はともかく，オブジェクトコードだけの場合はどうしてもディスアセンブラが必要になってきます．しかし，普通のディスアセンブラだと，プログラムの中にデータがあったりすると，途端にめちゃくちゃな結果を吐き出してくれるんですよね．

このセミオート・ディスアセンブラの場合，ある程度プログラムとデータを判別してくれるので，かなり正確にディスアセンブル・リストを出力してくれます．

このソフトは**I/Oの1987年11月号**を見て入力するか，**コムパックのディスケットサービス**を使って入手できます(ディスケットサービスの場合はソース付)．

● Tiny make

一般に広く使われている分割コンパイル/アセンブル支援ツール・makeのTiny版です．Tinyというのも，MS-DOSのMASMに付属する簡易版のmakeと同程度の機能しか用意していないからです．

本当はもっと高機能なmakeも出回ってはいますが，とりあえずこれでもお役に立つことと思います．

なお，このソフトは私が作りました．PDSとしてTeleStarなどに登録してありますので，自由にお使いください(ソース付)．

4章 浮動小数点演算パッケージ

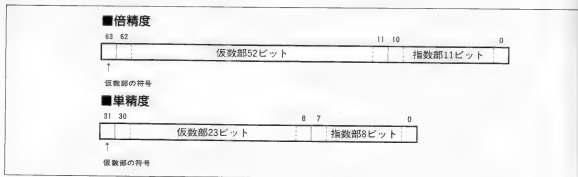
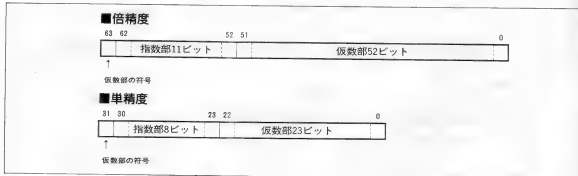
Human68k には浮動小数点演算機能をサポートするために"FLOAT1.X", "FLOAT2.X", そして"FLOAT3.X"の3種類の浮動小数点演算パッケージが用意されています.

これらのパッケージは特殊なデバイス・ドライバの形で Human68k に組み込まれます. 組み込んだ後は, 68000MPU の未定義命令, $FExx(xx は$00〜$FF)が有効となり, 簡単に浮動小数点演算を行なうことが可能になります. これを FE ファンクションコールと呼びます.

"FLOAT1.X", "FLAOT2.X"は, ソフトウェアで演算を行ないます. この2つの違いは浮動小数点数の表現方法の違いで, 前者は「シャープ・フォーマット」, 後者は「IEEE フォーマット」を採用しています. "FLOAT3.X"は別売りの数値演算プロセッサボード (CZ-6BP1) で演算を行ないます.

4.1 浮動小数点データ形式

浮動小数点数の形式は, 「シャープ・フォーマット」と「IEEEフォーマット」の2種類があり, さらに32ビット (4バイト) で表現される単精度, 64ビット (8バイト) で表現される倍精度の2種類に分けられます.

シャープ・フォーマット (FLOAT1.X)

IEEE フォーマット (FLOAT2.X, FLOAT3.X)

4.2 FE ファンクション・コールの呼び出し方

FEファンクションコールの呼び出しは，一般的に以下のような手順で行ないます．

① 各FEファンクションに渡すための値をレジスタに入れる（FEファンクションによってはユーザースタックに積む）．
② 未定義命令$FEに続くFEファンクション番号（1バイト）で構成されるFEファンクションコール命令を実行する．
③ FEファンクションによっては，戻ってきた後にユーザースタック・ポインタを補正．

FEファンクションを使う上では次の事項に注意してください．

① 引数を渡すために使ったレジスタ，返り値が入るレジスタ，ステータス・レジスタ（フラグ類）の内容は，FEファンクションコール実行後破壊される場合があります．必要な値の入ったレジスタは待避させておいてください．
② 返り値の返し方は統一されていません．スタックを利用して返すなど特殊な場合もありますので，各FEファンクションの解説の返り値の項で確認しておくことをお勧めします．
③ Human68kのファンクションコールと同様に，あらかじめFEファンクションコール用マクロを定義しておくと便利です．これは，その一例です．

```
FEFNC    macro    number
         dc. b    $fe
         dc. b    number
         endm
```

シャープからも"FEFUNC. H"というアセンブラ用のヘッダ・ファイルが提供されています．このファイルをアセンブラ・ソースの先頭でインクルードしておけば，"FPACK __UMUL" という具合いにFEファンクション名で呼び出すことができます．

"FEFUNC. H"は，Cコンパイラのライブラリ・ディスクの中の"CLIB. ARC"にアーカイブされていますので，AR. Xを使って取り出しておくと便利です．

数値演算プロセッサボードをお持ちの方は，付属ディスクの中のディレクトリ"¥SAMPLE"のなかに納められていますので，そちらを利用するとよいでしょう．

4.3 個々のFEファンクション使用解説

解説の例

$FE1B　　　　DTOL
↑　　　　　　↑
（FEファンクションコール命令）（FEファンクションコール名）

引数（引数の名前と型）
D0/D1　倍精度浮動小数点数
(D0が上位4バイト，D1が下位4バイトという意味)
返り値（返り値の入るレジスタ，型，そしてその意味）
D0. L　変換されたロングワード符号付き整数
コンディションコード（変化するフラグとその意味）
C　エラーがあればセット

フラグの名称には次の略称を使います
C:キャリーフラグ　Z:ゼロ・フラグ
X:エクステンドフラグ　V:オーバーフローフラグ
N:ネガティブフラグ

機能（機能の概略）
倍精度浮動小数点数をロングワード符号付き整数に変換します．少数部分は切り捨てられます．
変換結果がロングワード符号付き整数の範囲を超えた場合はエラーになります．

$FE00　__LMUL

引数

D0.L　被乗数

D1.L　乗数

返り値

D0.L　演算結果

コンディションコード

C　エラーがあればセット

機能

D0=D0×D1

ロングワード符号付き整数どうしの乗算を行ないます.

演算結果がロングワード符号付き整数を超えた場合はエラーとなります.

$FE01　__LDIV

引数

D0.L　被除数

D1.L　除数

返り値

D0.L　演算結果

コンディションコード

C　エラーがあればセット

機能

D0=D0÷D1

ロングワード符号付き整数どうしの除算を行ないます.

除数が0の場合はエラーとなります.

$FE02　__LMOD

引数

D0.L　被除数

D1.L　除数

返り値

D0.L　演算結果

コンディションコード

C　エラーがあればセット

機能

D0＝D0 mod D1

ロングワード符号付き整数どうしの除算の剰余を計算します.

除数が0の場合はエラーとなります.

$FE03　__UMUL

引数

D0.L　被乗数

D1.L　乗数

返り値

D0.L　演算結果

コンディションコード

C　エラーがあればセット

機能

D0＝D0×D1

ロングワード符号なし整数どうしの乗算を行ないます.

演算結果がロングワード符号なし整数を超えた場合はエラーとなります.

$FE04　__UDIV

引数

D0.L　被除数

D1.L　除数

返り値

D0.L　演算結果

コンディションコード

エラーがあればセット

機能

D0＝D0÷D1

ロングワード符号なし整数どうしの除算を行ないます．

除数が0の場合はエラーとなります．

$FE06　__UMOD

引数

D0.L　被除数

D1.L　除数

返り値

D0.L　演算結果

コンディションコード

エラーがあればセット

機能

D0＝D0 mod D1

ロングワード符号なし整数どうしの除算の剰余を計算します．

除数が0の場合はエラーとなります．

$FE08　__IMUL

引数

D0.L　被乗数

D1.L　乗数

返り値

D0.L　演算結果の上位4バイト

D1.L　演算結果の下位4バイト

コンディションコード

なし

機能

D0/D1＝D0×D1

ロングワード符号なし整数どうしの乗算を行ないます．演算結果は64ビットまで対応していますから，どのようなロングワード整数どうしを乗算してもエラーは発生しません．

$FE09　__IDIV

引数

D0.L　被除数

D1.L　除数

返り値

D0.L　演算結果（商）

D1.L　演算結果（剰余）

コンディションコード

C　エラーがあればセット

機能

D0＝D0÷D1（余りはD1へ）

ロングワード符号なし整数どうしの除算を行ないます．除数が0の場合はエラーとなります．

$FE0C　__RANDOMIZE

引数

D0.L　ランダム・シード　ロングワード符号付き整数

返り値

なし

コンディションコード

なし

機能

−32,768から32,767までの範囲でランダム・シードを与え，乱数を初期化します．引数の範囲が正しくない場合は乱数を初期化しません．

$FE0D　__SRAND

引数

D0.L　ランダム・シード　ロングワード符号付き整数

返り値

なし

コンディションコード

なし

機能

0から65,535までの範囲でランダム・シードを与え，乱数を初期化します．引数の範囲が正しくない場合は乱数を初期化しません．

$FE0E　__RAND

引数

なし

返り値

D0.L　乱数　ロングワード符号付き整数（0～32,767)

コンディションコード

なし

機能

ロングワード符号付き整数の乱数を返します．

$FE10　__STOL

引数

A0.L　文字列を指すポインタ

返り値

D0.L　変換されたロングワード符号付き整数

コンディションコード

C　エラーがあればセット

N　(Cがセットされている場合)数値の記述法がおかしい場合セット

V　(Cがセットされている場合)オーバーフローが発生した場合セット

機能

文字列をロングワード符号付き整数に変換します．文字列の最後にはNULLコードが必要です．

$FE11　__LTOS

引数

D0.L　ロングワード符号付き整数

A0.L　変換された文字列の格納用バッファを指すポインタ

返り値

(A0)～　変換された文字列

コンディションコード

なし

機能

ロングワード符号付き整数を文字列に変換します．バッファは充分な大きさをとっておいてください．

$FE12　　__STOH

引数

A0.L　文字列を指すポインタ

返り値

D0.L　変換されたロングワード符号なし整数

コンディションコード

C　エラーがあればセット

N　(Cがセットされている場合) 数値の記述法がおかしい場合セット

V　(Cがセットされている場合) オーバーフローが発生した場合セット

機能

16進数を表わす文字列をロングワード符号なし整数に変換します．文字列の最後にはNULLコードが必要です．

$FE13　　__HTOS

引数

A0.L　変換された文字列の格納用バッファを指すポインタ

D0.L　ロングワード符号なし整数

返り値

(A0)～　変換された文字列

コンディションコード

なし

機能

ロングワード符号なし整数を16進表現の文字列に変換します．バッファは充分な大きさをとっておいてください．

$FE14　　__STOO

引数

A0.L　文字列を指すポインタ

返り値

D0.L　変換されたロングワード符号なし整数

コンディションコード

C　エラーがあればセット

N　(Cがセットされている場合) 数値の記述法がおかしい場合セット

V　(Cがセットされている場合) オーバーフローが発生した場合セット

機能

8進数を表わす文字列をロングワード符号なし整数に変換します．

$FE15　　__OTOS

引数

A0.L　変換された文字列の格納用バッファを指すポインタ

D0.L　ロングワード符号なし整数

返り値

(A0)～　変換された文字列

コンディションコード

なし

機能

ロングワード符号なし整数を8進表現の文字列に変換します．

$FE16 __STOB

引数

A0.L　文字列を指すポインタ

返り値

D0.L　変換されたロングワード符号なし整数

コンディションコード

C　エラーがあればセット

N　(Cがセットされている場合)数値の記述法がおかしい場合セット

V　(Cがセットされている場合)オーバーフローが発生した場合セット

機能

2進数を表わす文字列をロングワード符号なし整数に変換します．

$FE17 __OTOB

引数

A0.L　変換された文字列の格納用バッファを指すポインタ

D0.L　ロングワード符号なし整数

返り値

(A0)～　変換された文字列

コンディションコード

なし

機能

ロングワード符号なし整数を2進表現の文字列に変換します．

$FE18 __IUSING

引数

D0.L　ロングワード符号付き整数

D1.L　桁数

A0.L　変換された文字列の格納用バッファを指すポインタ

返り値

(A0)～　変換された文字列

コンディションコード

なし

機能

ロングワード符号付き整数を文字列に変換します．変換後の数値が指定した桁数に満たない場合は，ゼロ・サプレスを行なった上で右詰めにして返します．

指定桁数よりも文字列が長くなる場合は，桁数を無視して必要なだけの長さの文字列を返します．

$FE1A __LTOD

引数

D0.L　ロングワード符号付き整数

返り値

D0/D1　変換された倍精度浮動小数点数

コンディションコード

なし

機能

ロングワード符号付き整数を倍精度浮動小数点数に変換します．

$FE1B　　　__DTOL

引数

D0/D1　倍精度浮動小数点数

返り値

D0.L　変換されたロングワード符号付き整数

コンディションコード

C　エラーがあればセット

機能

倍精度浮動小数点数をロングワード符号付き整数に変換します．小数部分は切り捨てられます．

変換結果がロングワード符号付き整数の範囲を超えた場合はエラーになります．

$FE1C　　　__LTOF

引数

D0.L　ロングワード符号付き整数

返り値

D0.L　変換された単精度浮動小数点数

コンディションコード

なし

機能

ロングワード符号付き整数を単精度浮動小数点数に変換します．

$FE1D　　　__FTOL

引数

D0.L　単精度浮動小数点数

返り値

D0.L　変換されたロングワード符号付き整数

コンディションコード

C　エラーがあればセット

機能

単精度浮動小数点数をロングワード符号付き整数に変換します．少数部分は切り捨てられます．

変換結果がロングワード符号付き整数の値の範囲を超えた場合はエラーになります．

$FE1E　　　__FTOD

引数

D0.L　単精度浮動小数点数

返り値

D0/D1　変換された倍精度浮動小数点数

コンディションコード

なし

機能

単精度浮動小数点数を倍精度浮動小数点数に変換します．

$FE1F　　　__DTOF

引数

D0/D1　倍精度浮動小数点数

返り値

D0.L　変換された単精度浮動小数点数

コンディションコード

C　エラーがあればセット

機能

倍精度浮動小数点数を単精度浮動小数点数に変換します．単精度浮動小数点数に変換できなかった場合，エラーとなります．

$FE20　　__VAL

引数

A0.L　文字列を指すポインタ

返り値

D0/D1　変換された倍精度浮動小数点数

D2.W　整数フラグ（文字列が10進数だった場合に有効）

D3.L　整数値（文字列が10進数だった場合に有効）

コンディションコード

C　エラーがあればセット

N　(Cがセットされているとき)数値の記述法がおかしい場合セット

V　(Cがセットされているとき)オーバーフローが発生した場合セット

機能

文字列を倍精度浮動小数点数に変換します．文字列の先頭に"&H"がついている場合は16進の"&O"がついている場合は8進の"&B"がついている場合は2進の表現されているものと判断します．

文字列が10進数だった場合，返り値としてD2とD3が有効となります．文字列が整数で，かつロングワード符号付き整数で表現可能な場合，整数フラグD2は$FFFFとなり，D3にはその整数値が入ります．それ以外の場合，整数フラグD2には0が入ります．

$FF21　　__USING

引数

D0/D1　倍精度浮動小数点数

D2.L　整数部分の桁数

D3.L　小数部分の桁数

D4.L　アトリビュート

A0.L　変換された文字列の格納用バッファを指すポインタ

返り値

(A0)～　変換された文字列

コンディションコード

なし

機能

倍精度浮動小数点数を文字列に変換します．文字列への変換はアトリビュートD4で指定した形で行ないます．

アトリビュートD4の各ビットは次のような意味を持っており，そのビットをセットすることにより，その機能が有効となります．複数のビットをセットしておくことは可能ですが，矛盾する指定を行なわないようにしてください．

bit0:　整数部分の指定桁数に満たない部分（左側）を"*"で埋める（通常はスペースで埋める）

bit1:　先頭に"¥"を付加する

bit2:　整数部分を3桁ごとに"," で区切る

bit3:　指数形式で表現する

bit4:　正の数の場合"＋"を先頭に付加する

bit5:　正の数の場合"＋"を，負の数の場合"-"を最後尾に付加する

bit6:　正の数の場合スペースを，負の数の場合"－"を最後尾に付加する

$FE22 __STOD

引数

A0.L 文字列を指すポインタ

返り値

D0/D1 変換された倍精度浮動小数点数

D2.W 整数フラグ

D3.L 整数値

コンディションコード

C エラーがあればセット

V (Cがセットされているとき)数値の記述法がおかしい場合セット

V (Cがセットされているとき)オーバーフローが発生した場合セット

機能

文字列を倍精度浮動小数点数に変換します．文字列が整数で，かつロングワード整数で表現可能な場合，整数フラグD2は$FFFFとなり，D3にはその整数値が入ります．それ以外の場合は整数フラグD2は0となります．

$FE23 __DTOS

引数

A0.L 変換された文字列の格納用バッファを指すポインタ

D0/D1 倍精度浮動小数点数

返り値

(A0)～ 変換された文字列

コンディションコード

なし

機能

倍精度浮動小数点数を文字列に変換します．

$FE24 __ECVT

引数

D0/D1 倍精度浮動小数点数

D2.B 全体の桁数

A0.L 変換された文字列の格納用バッファを指すポインタ

返り値

D0.L 小数点の位置

D1.L 符号 (0:正, 1:負)

(A0)～ 変換された文字列

コンディションコード

なし

機能

倍精度浮動小数点数を全体の桁数を指定して文字列に変換します．

例) D0/D1＝3.1415

D2.B ＝10(10進)

A0.L ＝result

という指定で__ECVTを呼び出した場合，次のような値が返されます．

```
result:  dc.b '3141500000',0
D0.L =1
D1.L =0
```

$FE25 __FCVT

引数

D0/D1 倍精度浮動小数点数

D2.B 小数点以下の桁数

A0.L 変換された文字列の格納用バッファを指すポインタ

返り値

D0.L 小数点の位置

D1.L 符号 (0:正, 1:負)

(A0)～ 変換された文字列

コンディションコード

なし

機能

倍精度浮動小数点数を，小数点以下の桁数を指定して文字列に変換します．返り値については$FE24__ECVTを参考にしてください．

$FE26 __GCVT

引数

D0/D1　倍精度浮動小数点数
D2.B　全体の桁数
A0.L　変換された文字列の格納用バッファを指すポインタ

返り値

(A0)～　変換された文字列

コンディションコード

なし

機能

倍精度浮動小数点数を，全体の桁数を指定した浮動小数点表記または指数表現の文字列に変換します．

負の値の場合は文字列の先頭に"－"が付加されます．通常は浮動小数点表記に変換されますが，桁数D2では表現できない場合に指数表現に変換されます．

$FE28 __DTST

引数

D0/D1　倍精度浮動小数点数

返り値

結果はフラグで返されまる

コンディションコード

Z　D0/D1が０ならばセット
N　D0/D1が負の数ならばセット

機能

D0/D1で与える倍精度浮動小数点数と０との比較をし，結果をフラグで返します．

$FE29 __DCMP

引数

D0/D1　被比較数
D2/D3　比較数

返り値

結果はフラグで返されます

コンディションコード

N　比較した結果が負であればセット
Z　比較した結果が０であればセット
C　ボローが発生した場合はセット

機能

倍精度浮動小数点数どうしを比較します．

被比較数D0/D1から比較数D2/D3を引いた結果がフラグにのみ返されます．フラグの結果によって，次のような関係を導き出すことができます．

D0/D1 > D2/D3
　C＝0, Z＝0, N＝0
D0/D1 ＝ D2/D3
　C＝0, Z＝1, N＝0
D0/D1 < D2/D3
　C＝1, Z＝0, N＝1

$FE2A __DNEG

引数

D0/D1　倍精度浮動小数点数値

返り値

D0/D1　演算結果

コンディションコード

なし

機能

倍精度浮動小数点数値の符号を反転します．

$FE2B __DADD

引数

D0/D1　被加算数

D2/D3　加算数

返り値

D0/D1　演算結果

コンディションコード

C　エラーがあればセット

V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合はクリア

機能

D0/D1＝D0/D1＋D2/D3

倍精度浮動小数点数どうしの加算を行ないます．

$FE2C __DSUB

引数

D0/D1　被減算数

D2/D3　減算数

返り値

D0/D1　演算結果

コンディションコード

C　エラーがあればセット

V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合はクリア

機能

D0/D1＝D0/D1－D2/D3

倍精度浮動小数点数どうしの減算を行ないます．

$FE2D __DMUL

引数

D0/D1　被乗数

D2/D3　乗数

返り値

D0/D1　演算結果

コンディションコード

C　エラーがあればセット

V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合はクリア

機能

D0/D1＝D0/D1×D2/D3

倍精度浮動小数点数どうしの乗算を行ないます．

$FE2E __DDIV

引数

D0/D1　被除数

D2/D3　除数

返り値

D0/D1　演算結果

コンディションコード

C　エラーがあればセット

Z　(Cがセットされている場合) 0で割ったときセット

V　(Cがセットされている場合) オーバーフローの場合セット，アンダーフローの場合はクリア

機能

D0/D1＝D0/D1÷D2/D3

倍精度浮動小数点数どうしの除算を行ないます．

$FE2F __DMOD

引数

D0/D1　被除数

D2/D3　除数

返り値

D0/D1　演算結果

コンディションコード

C　エラーがあればセット

Z　(Cがセットされているとき) 0で割ったときセット

V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合はクリア

機能

D0/D1＝D0/D1modD2/D3

倍精度浮動小数点数どうしの剰余を求めます．

$FE30 __DABS

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　演算結果

コンディションコード

なし

機能

倍精度浮動小数点数の絶対値を求めます．

$FE31 __DCEIL

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　演算結果

コンディションコード

なし

機能

D0/D1で与えた倍精度浮動小数点数と等しいか，それ以上の最小の整数を返します．

$FE32 __DFIX

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　演算結果

コンディションコード

なし

機能

D0/D1で与えた倍精度浮動小数点数の整数部を求めます．

$FE33 __DFLOOR

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　演算結果

コンディションコード

なし

機能

D0/D1で与えた倍精度浮動小数点数と等しいかまたはそれより小さい最大の整数を返します．

$FE34　　__DFRAC

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　演算結果

コンディションコード

なし

機能

D0/D1で与えた倍精度浮動小数点数の小数部を求めます．

$FE35　　__DSGN

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　結果（倍精度浮動小数点数）

コンディションコード

なし

機能

D0/D1で与えた倍精度浮動小数点数が正か負か0かを調べます．

正なら＋1，負なら－1，0なら0を返します．

$FE36　　__SIN

引数

D0/D1　角度（ラジアン単位，倍精度浮動小数点数）

返り値

D0/D1　演算結果

コンディションコード

なし

機能

角度（ラジアン単位）を与えてsinを計算します．

$FE37　　__COS

引数

D0/D1　角度（ラジアン単位，倍精度浮動小数点数）

返り値

D0/D1　演算結果

コンディションコード

なし

機能

角度（ラジアン単位）を与えてcosを計算します．

$FE38　　__TAN

引数

D0/D1　角度（ラジアン単位，倍精度浮動小数点数）

返り値

D0/D1　演算結果

コンディションコード

C　　エラーがあればセット

機能

角度（ラジアン単位）を与えてtanを計算します．結果が無限大になるような角度を与えた場合，エラーになります．

$FE39　　__ATAN

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　演算結果（ラジアン単位）

コンディションコード

なし

機能

atanを計算します．

$FE3A　　__LOG

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　演算結果

コンディションコード

C　エラーがあればセット

Z　(Cがセットされているとき)引数が0の場合セット

機能

logを計算します．引数が0の場合エラーになります．

$FE3B　　__EXP

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　演算結果

コンディションコード

C　エラーがあればセット

機能

指数関数を計算します．オーバーフローが発生した場合エラーとなります．

$FE3C　　__SQR

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　演算結果

コンディションコード

C　エラーがあればセット

Z　(Cがセットされている場合)引数が0の場合セット

機能

平方根を計算します．引数が負の場合エラーになります．

$FE3D　　__PI

引数

なし

返り値

D0/D1　π

コンディションコード

なし

機能

πを倍精度浮動小数点数の範囲内で返します．

$FE3E　　　__NPI

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　演算結果

コンディションコード

C　エラーがあればセット

機能

π の引数倍の値を倍精度浮動小数点数の範囲内で返します．オーバーフローが発生した場合，エラーになります．

$FE3F　　　__POWER

引数

D0/D1　被べき乗数

D2/D3　べき乗数

返り値

D0/D1　演算結果

コンディションコード

C　エラーがあればセット

V　(Cがセットされている場合)オーバーフローの場合セット，アンダーフローの場合クリア

機能

$D0/D1=D0/D1^{D2/D3}$

べき乗を計算します.オーバーフロー/アンダーフローが発生した場合はエラーになります.

$FE40　　　__RND

引数

なし

返り値

D0/D1　乱数(倍精度浮動小数点数)

コンディションコード

なし

機能

倍精度浮動小数点数で乱数(0以上1未満)を返します.

$FE49　　　__DFREXP

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　仮数部を示す倍精度浮動小数点数

D2.L　指数部を示すロングワード整数

コンディションコード

なし

機能

倍精度浮動小数点数の仮数部と指数部を分けます.

返り値D0/D1は，引数の仮数部はそのままで，指数部を1に変えたものを返します.

$FE4A　　　__DLDEXP

引数

D0/D1　仮数部データ（倍精度浮動小数点数）

D2.L　指数部データ（ロングワード符号付き整数）

返り値

D0/D1　引数を結合した倍精度浮動小数点数

コンディションコード

C　エラーがあればセット

機能

引数の仮数部データD0/D1と指数部データD2を結合した倍精度浮動小数点数を返します.

$FE4B　__DADDONE

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　演算結果

コンディションコード

なし

機能

引数の倍精度浮動小数点数に1を加えます．

$FE4C　__DSUBONE

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　演算結果

コンディションコード

なし

機能

引数の倍精度浮動小数点数から1を引きます．

$FE4D　__DDIVTWO

引数

D0/D1　倍精度浮動小数点数

返り値

D0/D1　演算結果

コンディションコード

C　エラーがあればセット

機能

引数の倍精度浮動小数点数を2で割ります．アンダーフローが発生した場合エラーになります．

$FE4E　__DIEECNV

引数

D0/D1　シャープフォーマット倍精度浮動小数点数

返り値

D0/D1　IEEEフォーマット倍精度浮動小数点数

コンディションコード

なし

機能

シャープフォーマット倍精度浮動小数点数をIEEEフォーマットの倍精度浮動小数点数に変換します．

FLOAT1.Xを使っている場合にのみ意味を持ちます．

$FE4F　__IEEDCNV

引数

D0/D1　IEEEフォーマット倍精度浮動小数点数

返り値

D0/D1　シャープフォーマット倍精度浮動小数点数

コンディションコード

なし

機能

IEEEフォーマット倍精度浮動小数点数をシャープフォーマット倍精度浮動小数点数に変換します．

FLOAT1.Xを使っている場合にのみ意味を持ちます．

$FE50　__FVAL

引数

A0.L　文字列を指すポインタ

返り値

D0.L　変換された単精度浮動小数点数

D2.W　整数フラグ（文字列が10進数だった場合に有効）

D3.L　整数値（文字列が10進数だった場合に有効）

コンディションコード

C　エラーがあればセット

N　(Cがセットされているとき)数値の記述法がおかしい場合セット

V　(Cがセットされている場合)オーバーフローが発生した場合セット

機能

文字列を単精度浮動小数点数に変換します．文字列の先頭に"&H"がついている場合は16進"&O"がついている場合は8進"&B"がついている場合は2進表現されているものと判断します．

文字列が10進数だった場合，返り値としてD2とD3が有効となります．文字列が整数で，かつロングワード符号付き整数で表現可能な場合，整数フラグD2は$FFFFとなり，D3にはその整数値が入ります．それ以外の場合，整数フラグD2には0が入ります．

$FE51　__FUSING

引数

D0.L　単精度浮動小数点数

D2.L　整数部分の桁数

D3.L　小数部分の桁数

D4.L　アトリビュート

A0.L　変換された文字列の格納用バッファを指すポインタ

返り値

(A0)～　変換された文字列

コンディションコード

なし

機能

単精度浮動小数点数を文字列に変換します．文字列への変換はアトリビュートD4で指定した形で行ないます．

アトリビュートの内容は次のとおりです．

bit0:　整数部分の指定桁数に満たない部分（左側）を"*"で埋める（通常はスペースで埋める）

bit1:　先頭に"¥"を付加する

bit2:　整数部分を3桁ごとに"，"で区切る

bit3:　指数形式で表現する

bit4:　正の数の場合"＋"を先頭に付加する

bit5:　正の数の場合"＋"を，負の数の場合"－"を最後尾に付加する

bit6:　正の数の場合"(スペース)"を，負の数の場合"－"を最後尾に付加する

$FE52　__STOF

引数

A0.L　文字列を指すポインタ

返り値

D0.L　変換された単精度浮動小数点数

D2.W　整数フラグ

D3.L　整数値

コンディションコード

C　エラーがあればセット

N　(Cがセットされている場合)数値の記述法がおかしい場合セット

V　(Cがセットされている場合)オーバーフローが発生すればセット

機能

文字列を単精度浮動小数点数に変換します．文字列が整数で，かつロングワード整数で表現可能な場合，整数フラグD2は$FFFFとなり，D3にはその整数値が入ります．

それ以外の場合は整数フラグD2は0となります．

$FE53 __FTOS

引数

A0.L　変換された文字列の格納用バッファを指すポインタ

D0/D1　単精度浮動小数点数

返り値

(A0)～　変換された文字列

コンディションコード

なし

機能

単精度浮動小数点数を文字列に変換します.

$FE54 __FECVT

引数

D0/D1　単精度浮動小数点数

D2.B　全体の桁数

A0.L　変換された文字列の格納用バッファを指すポインタ

返り値

D0.L　小数点の位置

D1.L　符号（0:正, 1:負）

(A0)～　変換された文字列

コンディションコード

なし

機能

単精度浮動小数点数を全体の桁数を指定して文字列に変換します. 返り値については$FE24 __ECVTを参考にしてください.

$FE55 __FFCVT

引数

D0/D1　単精度浮動小数点数

D2.B　小数点以下の桁数

A0.L　変換された文字列の格納用バッファを指すポインタ

返り値

D0.L　小数点の位置

D1.L　符号（0:正, 1:負）

(A0)～　変換された文字列

コンディションコード

なし

機能

単精度浮動小数点数を小数点以下の桁数を指定して文字列に変換します.

返り値については$FE24 __ECVTを参考にしてください.

$FE56 __FGCVT

引数

D0.L　単精度浮動小数点数

D2.B　全体の桁数

A0.L　変換された文字列の格納用バッファを指すポインタ

返り値

(A0)～　変換された文字列

コンディションコード

なし

機能

単精度浮動小数点数を全体の桁数を指定した浮動小数点表現または指数表現の文字列に変換します.

負の値の場合は文字列の先頭に"－"が付加されます. 通常は浮動小数点表記に変換されますが, 桁数D2では表現できない場合に指数表現に変換されます.

$FE58 __FTST

引数

D0.L　単精度浮動小数点数

返り値

結果はフラグで返されます

コンディションコード

Z　D0/D1が0ならばセット

N　D0/D1が負の数ならばセット

機能

D0/D1で与える単精度浮動小数点数と0との比較をし，結果をフラグで返します．

$FE59 __FCMP

引数

D0.L　被比較数

D1.L　比較数

返り値

結果はフラグで返されます

コンディションコード

N　比較した結果が負であればセット

Z　比較した結果が0であればセット

C　ボローが発生した場合はセット

機能

単精度浮動小数点数どうしを比較します．

被比較数D0から比較数D1を引いた結果がフラグにのみ返されます．フラグの結果によって，次のような関係を導き出すことができます．

D0 > D1

　C=0, Z=0, N=0

D0 = D1

　C=0, Z=1, N=0

D0 < D1

　C=1, Z=0, N=1

$FE5A __FNEG

引数

D0.L　単精度浮動小数点数値

返り値

D0.L　演算結果

コンディションコード

なし

機能

単精度浮動小数点数値の符号を反転します．

$FE5B __FADD

引数

D0.L　被加算数

D1.L　加算数

返り値

D0/D1　演算結果

コンディションコード

C　エラーがあればセット

V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合はクリア

機能

D0=D0+D1

単精度浮動小数点数どうしの加算を行ないます．

$FE5C　　__FSUB

引数

D0.L　被減算数

D1.L　減算数

返り値

D0/D1　演算結果

コンディションコード

C　エラーがあればセット

V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合はクリア

機能

D0＝D0－D1

単精度浮動小数点数どうしの減算を行ないます.

$FE5D　　__FMUL

引数

D0.L　被乗数

D1.L　乗数

返り値

D0.L　演算結果

コンディションコード

C　エラーがあればセット

V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合はクリア

機能

D0＝D0×D1

単精度浮動小数点数どうしの乗算を行ないます.

$FE5E　　__FDIV

引数

D0.L　被除数

D1.L　除数

返り値

D0.L　演算結果

コンディションコード

C　エラーがあればセット

Z　(Cがセットされているとき) 0で割ったときセット

V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合はクリア

機能

D0＝D0÷D1

単精度浮動小数点数どうしの除算を行ないます.

$FE5F　　__FMOD

引数

D0.L　被除数

D1.L　除数

返り値

D0.L　演算結果

コンディションコード

C　エラーがあればセット

Z　(Cがセットされているとき) 0で割ったときセット

V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合はクリア

機能

D0＝D0 mod D1

単精度浮動小数点数どうしの剰余を求めます.

$FE60　　__FABS

引数

D0.L　単精度浮動小数点数

返り値

D0.L　演算結果

コンディションコード

なし

機能

単精度浮動小数点数の絶対値を求めます.

$FE61　　__FCEIL

引数

D0.L　単精度浮動小数点数

返り値

D0.L　演算結果

コンディションコード

なし

機能

D0で与えた単精度浮動小数点数と等しいか, それ以上の最小の整数を返します.

$FE62　　__FFIX

引数

D0.L　単精度浮動小数点数

返り値

D0.L　演算結果

コンディションコード

なし

機能

D0で与えた単精度浮動小数点数の整数部を求めます.

$FE63　　__FFLOOR

引数

D0.L　単精度浮動小数点数

返り値

D0.L　演算結果

コンディションコード

なし

機能

D0で与えた単精度浮動小数点数と等しいかまたはそれより小さい最大の整数を返します.

$FE64　　__FFRAC

引数

D0.L　単精度浮動小数点数

返り値

D0.L　演算結果

コンディションコード

なし

機能

D0で与えた単精度浮動小数点数の小数部を求めます.

$FE65　　__FSGN

引数

D0.L　単精度浮動小数点数

返り値

D0.L　結果(単精度浮動小数点数)

コンディションコード

なし

機能

D0で与えた単精度浮動小数点数が正か負か0かを調べます.

正なら+1, 負なら−1, 0なら0を返します.

$FE66　　　__FSIN

引数

D0.L　角度（ラジアン単位，単精度浮動小数点数）

返り値

D0.L　演算結果

コンディションコード

なし

機能

角度（ラジアン単位）を与えてsinを計算します．

$FE67　　　__FCOS

引数

D0.L　角度（ラジアン単位，単精度浮動小数点数）

返り値

D0.L　演算結果

コンディションコード

なし

機能

角度（ラジアン単位）を与えてcosを計算します．

$FE68　　　__FTAN

引数

D0.L　角度（ラジアン単位，単精度浮動小数点数）

返り値

D0.L　演算結果

コンディションコード

C　エラーがあればセット

機能

角度（ラジアン単位）を与えてtanを計算します．結果が無限大になるような角度を与えた場合，エラーになります．

$FE69　　　__FATAN

引数

D0.L　単精度浮動小数点数

返り値

D0.L　演算結果（ラジアン単位）

コンディションコード

なし

機能

atanを計算します．

$FE6A　　　__FLOG

引数

D0.L　単精度浮動小数点数

返り値

D0.L　演算結果

コンディションコード

C　エラーがあればセット

Z　（Cがセットされているとき）引数が0の場合セット

機能

logを計算します．引数が0の場合エラーになります．

$FE6B __FEXP

引数

D0.L 単精度浮動小数点数

返り値

D0.L 演算結果

コンディションコード

C エラーがあればセット

機能

指数関数を計算します．オーバーフローが発生した場合エラーとなります．

$FE6C __FSQR

引数

D0.L 単精度浮動小数点数

返り値

D0.L 演算結果

コンディションコード

C エラーがあればセット

Z (Cがセットされている場合)引数が0の場合セット

機能

平方根を計算します．引数が負の場合エラーになります．

$FE6D __FPI

引数

なし

返り値

D0.L π

コンディションコード

なし

機能

πを単精度浮動小数点数の範囲内で返します．

$FE6E __FNPI

引数

D0.L 単精度浮動小数点数

返り値

D0.L 演算結果

コンディションコード

C エラーがあればセット

機能

πの引数倍の値を単精度浮動小数点数の範囲内で返します．

オーバーフローが発生した場合，エラーになります．

$FE6F __FPOWER

引数

D0.L 被べき乗数

D1.L べき乗数

返り値

D0.L 演算結果

コンディションコード

C エラーがあればセット

V (Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合クリア

機能

$D0 = D0^{D1}$

べき乗を計算します．オーバーフロー/アンダーフローが発生した場合はエラーになります．

$FE70　　__FRND

引数

なし

返り値

D0.L　乱数（単精度浮動小数点数）

コンディションコード

なし

機能

単精度浮動小数点数で乱数（0以上1未満）を返します.

$FE79　　__FFREXP

引数

D0.L　単精度浮動小数点数

返り値

D0.L　仮数部を示す単精度浮動小数点数

D1.L　指数部を示すロングワード整数

コンディションコード

なし

機能

単精度浮動小数点数の仮数部と指数部を分けます.

返り値D0/D1は, 引数の仮数部はそのままで, 指数部を1に変えたものを返します.

$FE7A　　__FLDEXP

引数

D0.L　仮数部データ（単精度浮動小数点数）

D1.L　指数部データ（ロングワード符号付き整数）

返り値

D0.L　引数を結合した単精度浮動小数点数

コンディションコード

C　エラーがあればセット

機能

引数の仮数部データD0と指数部データD1を結合した単精度浮動小数点数を返します.

$FE7B　　__FADDONE

引数

D0.L　単精度浮動小数点数

返り値

D0.L　演算結果

コンディションコード

なし

機能

引数の単精度浮動小数点数に1を加えます.

$FE7C　　__FSUBONE

引数

D0.L　単精度浮動小数点数

返り値

D0.L　演算結果

コンディションコード

なし

機能

引数の単精度浮動小数点数から1を引きます.

$FE7D　　__FDIVTWO

引数

D0.L　単精度浮動小数点数

返り値

D0.L　演算結果

コンディションコード

C　エラーがあればセット

機能

引数の単精度浮動小数点数を2で割ります．

アンダーフローが発生した場合エラーになります．

$FE7E　　__FIEECNV

引数

D0.L　シャープフォーマット単精度浮動小数点数

返り値

D0.L　IEEEフォーマット単精度浮動小数点数

コンディションコード

なし

機能

シャープフォーマット単精度浮動小数点数をIEEEフォーマットに変換します．FLOAT1.Xを使っている場合にのみ意味を持ちます．

$FE7F　　__IEEFCNV

引数

D0.L　IEEEフォーマット単精度浮動小数点数

返り値

D0.L　シャープフォーマット単精度浮動小数点数

コンディションコード

なし

機能

IEEEフォーマット単精度浮動小数点数をシャープフォーマット単精度浮動小数点数に変換します．FLOAT1.Xを使っている場合にのみ意味を持ちます．

$FEE0　　__CLMUL

引数

(A7).L　被乗数のロングワード符号付き整数

4(A7).L　乗数のロングワード符号付き整数

返り値

(A7).L　演算結果のロングワード符号付き整数

コンディションコード

C　エラーがあればセット

機能

(A7) = (A7) × 4 (A7)

ロングワード符号付き整数どうしの乗算を行ないます．演算結果がロングワード符号付き整数の範囲を超えた場合はエラーとなります．

$FEE1　　__CLDIV

引数

(A7).L　被除数のロングワード符号付き整数

4(A7).L 除数のロングワード符号付き整数

返り値

(A7).L　演算結果のロングワード符号付き整数

コンディションコード

C　　エラーがあればセット

機能

(A7) = (A7) ÷ 4 (A7)

ロングワード符号付き整数どうしの除算を行ないます．除数4(A7)が0の場合はエラーとなります．

$FEE2　　__CLMOD

引数

(A7).L　被除数のロングワード符号付き整数

4(A7).L 除数のロングワード符号付き整数

返り値

(A7).L　演算結果のロングワード符号付き整数

コンディションコード

C　　エラーがあればセット

機能

(A7) = (A7) mod 4 (A7)

ロングワード符号付き整数どうしの除算の剰余を求めます．除数4(A7)が0の場合はエラーとなります．

$FEE3　　__CUMUL

引数

(A7).L　被乗数のロングワード符号なし整数

4(A7).L 乗数のロングワード符号なし整数

返り値

(A7).L　演算結果のロングワード符号なし整数

コンディションコード

C　　エラーがあればセット

機能

(A7) = (A7) × 4 (A7)

ロングワード符号なし整数どうしの乗算を行ないます．演算結果がロングワード符号なし整数の範囲を超えた場合はエラーとなります．

$FEE4　　__CUDIV

引数

(A7).L　被除数のロングワード符号なし整数

4(A7).L 除数のロングワード符号なし整数

返り値

(A7).L　演算結果のロングワード符号なし整数

コンディションコード

C　　エラーがあればセット

機能

(A7) = (A7) ÷ 4 (A7)

ロングワード符号なし整数どうしの除算を行ないます．除数4(A7)が0の場合はエラーとなります．

$FEE5　　__CUMOD

引数

(A7).L　被除数のロングワード符号なし整数
4(A7).L 除数のロングワード符号なし整数

返り値

(A7).L　演算結果のロングワード符号なし整数

コンディションコード

C　　エラーがあればセット

機能

(A7) = (A7) mod 4 (A7)

ロングワード符号なし整数どうしの除算の剰余を求めます．除数4(A7)が0の場合はエラーとなります．

$FEE6　　__CLTOD

引数

(A7).L　ロングワード符号付き整数

返り値

(A7)/4(A7)　倍精度浮動小数点数

コンディションコード

なし

機能

ロングワード符号付き整数を倍精度浮動小数点数に変換します．引数は4バイト，返り値は8バイトであることに注意してください．

例）実際には次のように使います．

```
move.l  #144, -(sp)
FEFNC   $E6          * __CLTOD
move.l  (sp), d0
move.l  4(sp), d1
addq.l  #4, sp
```

以上の操作でD0/D1に返り値が入ります．

$FEE7　　__CDTOL

引数

(A7)/4(A7)　倍精度浮動小数点数

返り値

(A7)　　変換されたロングワード符号付き整数

コンディションコード

C　　エラーがあればセット

機能

倍精度浮動小数点数をロングワード符号付き整数に変換します．小数部分は切り捨てられます．エラーは変換結果がロングワード符号付き整数の範囲を超えた場合に生じます．引数は8バイト，返り値は4バイトであることに注意してください．

例）実際には次のように使います．
D0/D1に倍精度浮動小数点数が入っている場合．

```
move.l  d1, -(sp)
move.l  d0, -(sp)
FEFNC   $E7          * __CDTOL
move.l  (sp), d0
addq.l  #8, sp
```

以上の操作でD0に返り値が入ります．

$FEE8　　__CLTOF

引数

(A7).L　ロングワード符号付き整数

返り値

(A7).L　変換された単精度浮動小数点数

コンディションコード

なし

機能

ロングワード符号付き整数を単精度浮動小数点数に変換します．

$FEE9 __CFTOL

引数

(A7).L　単精度浮動小数点数

返り値

(A7).L　変換されたロングワード符号付き整数

コンディションコード

C　　エラーがあればセット

機能

単精度浮動小数点数をロングワード符号付き整数に変換します．小数部分は切り捨てられます．エラーは変換結果がロングワード符号付き整数の範囲を超えた場合に生じます．

$FEEA __CFTOD

引数

(A7).L　単精度浮動小数点数

返り値

(A7)/4(A7)　変換された倍精度浮動小数点数

コンディションコード

なし

機能

単精度浮動小数点数を倍精度浮動小数点数に変換します．引数は4バイト，返り値は8バイトであることに注意してください．

例） 実際には次のように使います．
D0に単精度浮動小数点数が入っている場合，

```
move.l  d0, -(sp)
FEFNC   $EA         * __CFTOD
move.l  (sp), d0
move.l  4(sp), d1
addq.l  #4, sp
```

以上の操作でD0/D1に返り値が入ります．

$FEEB __CDTOF

引数

(A7)/4(A7)　倍精度浮動小数点数

返り値

(A7).L　変換された単精度浮動小数点数

コンディションコード

C　　エラーが発生した場合セット

機能

倍精度浮動小数点数を単精度浮動小数点数に変換します．エラーは引数が単精度浮動小数点数で表現できない場合に生じます．

例） 実際には次のように使います．
D0/D1に倍精度浮動小数点数が入っている場合，

```
move.l  d1, -(sp)
move.l  d0, -(sp)
FEFNC   $EB         * __CDTOF
move.l  (sp), d0
addq.l  #8, sp
```

以上の操作でD0に返り値が入ります．

$FEEC　　　__CDCMP

引数

(A7)/4(A7)　被比較数の倍精度浮動小数点数
8(A7)/12(A7)　比較数の倍精度浮動小数点数

返り値

結果はフラグで返されます

コンディションコード

N　比較した結果が負であればセット
Z　比較した結果が0であればセット
C　ボローが発生した場合はセット

機能

倍精度浮動小数点数どうしを比較します．

被比較数(A7)/4(A7)から比較数8(A7)/12(A7)を引いた結果がフラグにのみ返されます．フラグの結果によって，次のような関係を導き出すことができます．

(A7)/4(A7)>8(A7)/12(A7)
　C=0, Z=0, N=0
(A7)/4(A7)=8(A7)/12(A7)
　C=0, Z=1, N=0
(A7)/4(A7)<8(A7)/12(A7)
　C=1, Z=0, N=1

$FEED　　　__CDADD

引数

(A7)/4(A7)　被加算数の倍精度浮動小数点数
8(A7)/12(A7)　加算数の倍精度浮動小数点数

返り値

(A7)/4(A7)　演算結果の倍精度浮動小数点数

コンディションコード

C　エラーがあればセット
V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合クリア

機能

(A7)/4(A7)=(A7)/4(A7)+8(A7)/12(A7)
倍精度浮動小数点数どうしの加算を行ないます．

$FEEE　　　__CDSUB

引数

(A7)/4(A7)　被減算数の倍精度浮動小数点数
8(A7)/12(A7)　減算数の倍精度浮動小数点数

返り値

(A7)/4(A7)　演算結果の倍精度浮動小数点数

コンディションコード

C　エラーがあればセット
V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合クリア

機能

(A7)/4(A7)=(A7)/4(A7)−8(A7)/12(A7)
倍精度浮動小数点数どうしの減算を行ないます．

$FEEF　　　__CDMUL

引数

(A7)/4(A7)　被乗算数の倍精度浮動小数点数
8(A7)/12(A7)　乗算数の倍精度浮動小数点数

返り値

(A7)/4(A7)　演算結果の倍精度浮動小数点数

コンディションコード

C　エラーがあればセット
V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合クリア

機能

(A7)A/4(A7)=(A7)/4(A7)×8(A7)/12(A7)
倍精度浮動小数点数どうしの乗算を行ないます．

$FEF0　　　__CDDIV

引数

(A7)/4(A7)　被除算数の倍精度浮動小数点数
8(A7)/12(A7)　除算数の倍精度浮動小数点数

返り値

(A7)/4(A7)　演算結果の倍精度浮動小数点数

コンディションコード

C　エラーがあればセット
Z　(Cがセットされているとき) 0で割ったときセット
V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合クリア

機能

(A7)/4(A7)＝(A7)/4(A7)÷8(A7)/12(A7)

倍精度浮動小数点数どうしの除算を行ないます．

$FEF1　　　__CDMOD

引数

(A7)/4(A7)　被除算数の倍精度浮動小数点数
8(A7)/12(A7)　除算数の倍精度浮動小数点数

返り値

(A7)/4(A7)　演算結果の倍精度浮動小数点数

コンディションコード

C　エラーがあればセット
Z　(Cがセットされているとき) 0で割ったときセット
V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合クリア

機能

(A7)/4(A7)＝(A7)/4(A7)mod8(A7)/12(A7)

倍精度浮動小数点数どうしの除算を行ない，その剰余を求めます．

$FEF2　　　__CFCMP

引数

(A7).L　被比較数の単精度浮動小数点数
4(A7).L　比較数の単精度浮動小数点数

返り値

結果はフラグで返されます

コンディションコード

N　比較した結果が負であればセット
Z　比較した結果が0であればセット
C　ボローが発生した場合はセット

機能

単精度浮動小数点数どうしを比較します．

被比較数(A7)から比較数4(A7)を引いた結果がフラグにのみ返されます．フラグの結果によって，次のような関係を導き出すことができます．

(A7)＞4(A7)
　C＝0, Z＝0, N＝0
(A7)＝4(A7)
　C＝0, Z＝1, N＝0
(A7)＜4(A7)
　C＝1, Z＝0, N＝1

$FEF3　　　__CFADD

引数

(A7).L　被加算数の単精度浮動小数点数
4(A7).L　加算数の単精度浮動小数点数

返り値

(A7)　演算結果の単精度浮動小数点数

コンディションコード

C　エラーがあればセット
V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合クリア

機能

(A7)＝(A7)＋4(A7)

単精度浮動小数点数どうしの加算を行ないます．

$FEF4 __CFSUB

引数

(A7).L　被減算数の単精度浮動小数点数

4(A7).L　減算数の単精度浮動小数点数

返り値

(A7).L　演算結果の単精度浮動小数点数

コンディションコード

C　エラーがあればセット

V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合クリア

機能

(A7) = (A7) − 4(A7)

単精度浮動小数点数どうしの減算を行ないます.

$FEF5 __CFMUL

引数

(A7).L　被乗算数の単精度浮動小数点数

4(A7).L　乗算数の単精度浮動小数点数

返り値

(A7).L　演算結果の単精度浮動小数点数

コンディションコード

C　エラーがあればセット

V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合クリア

機能

(A7) = (A7) × 4(A7)

単精度浮動小数点数どうしの乗算を行ないます.

$FEF6 __CFDIV

引数

(A7).L　被除算数の単精度浮動小数点数

4(A7).L　除算数の単精度浮動小数点数

返り値

(A7).L　演算結果の単精度浮動小数点数

コンディションコード

C　エラーがあればセット

Z　(Cがセットされているとき) 0で割ったときセット

V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合クリア

機能

(A7) = (A7) ÷ 4(A7)

単精度浮動小数点数どうしの除算を行ないます.

$FEF7 __CFMOD

引数

(A7).L　被除算数の単精度浮動小数点数

4(A7).L　除算数の単精度浮動小数点数

返り値

(A7).L　演算結果の単精度浮動小数点数

コンディションコード

C　エラーがあればセット

Z　(Cがセットされているとき) 0で割ったときセット

V　(Cがセットされているとき)オーバーフローの場合セット，アンダーフローの場合クリア

機能

(A7) = (A7) mod4(A7)

単精度浮動小数点数どうしの除算を行ない，その剰余を求めます.

$FEF8 __CDTST

引数

(A7)/4(A7)　倍精度浮動小数点数

返り値

結果はフラグで返されます

コンディションコード

Z　(A7)/4(A7)が0ならばセット

N　(A7)/4(A7)が負の数ならセット

機能

(A7)/4(A7)で与える倍精度浮動小数点数と0との比較をし，結果をフラグで返します．

$FEF9 __CFTST

引数

(A7).L　単精度浮動小数点数

返り値

結果はフラグで返されます

コンディションコード

Z　(A7)が0ならばセット

N　(A7)が負の数ならセット

機能

(A7)で与える単精度浮動小数点数と0との比較をし，結果をフラグで返します．

$FEFA __CDINC

引数

(A7)/4(A7)　倍精度浮動小数点数

返り値

(A7)/4(A7)　演算結果の倍精度浮動小数点数

コンディションコード

なし

機能

引数の倍精度浮動小数点数に1を加えます．

$FEFB __CFINC

引数

(A7).L　単精度浮動小数点数

返り値

(A7).L　演算結果の単精度浮動小数点数

コンディションコード

なし

機能

引数の単精度浮動小数点数に1を加えます．

$FEFC __CDDEC

引数

(A7)/4(A7)　倍精度浮動小数点数

返り値

(A7)/4(A7)　演算結果の倍精度浮動小数点数

コンディションコード

なし

機能

引数の倍精度浮動小数点数から1を引きます．

$FEFD　　　__CFDEC

引数

(A7).L　単精度浮動小数点数

返り値

(A7).L　演算結果の単精度浮動小数点数

コンディションコード

なし

機能

引数の単精度浮動小数点数から1を引きます.

$FEFE　　　__FEVARG

引数

なし

返り値

D0.L　　数値ドライバ名(1)

D1.L　　数値ドライバ名(2)

コンディションコード

なし

機能

組み込まれている数値演算デバイス・ドライバの種類を調べます.

FLOAT1.X:	D0.L	'HS86'	($48533836)
	D1.L	'SOFT'	($534F4654)
FLOAT2.X:	D0.L	'IEEE'	($49454545)
	D1.L	'SOFT'	($534F4654)
FLOAT3.X:	D0.L	'IEEE'	($49454545)
	D1.L	'FPCP'	($46504350)

$FEFF　　　__FEVECS

引数

D0.L　　FETBLの番号 ($FE00～$FEFE)

A0.L　　処理アドレス

返り値

D0.L　　前の処理アドレス

コンディションコード

なし

機能

D0で指定したFEファンクションの処理先アドレスを変更します. D0に範囲外の番号を入れた場合, エラーとなり, D0には－1が返ってきます.

```
*
*       macro defination
*
FNC             macro   number
                dc.b    $ff,number
                endm
IOCS            macro   number
                move.l  #number,d0
                trap    #15
                endm
```

```
PUSH            macro   reg_list
                movem.l reg_list,-(sp)
                endm
POP             macro   reg_list
                movem.l (sp)+,reg_list
                endm

TRUE            =       -1
FALSE           =       0
```

```
*
*       FEFUNC sample program
*

                include macro.h

FEFUNC          macro   number
                dc.b    $FE
                dc.b    number
                endm

                .text
start:
                tst.b   (A2)+           *コマンドライン の文字数チェック
                beq     noarg           *   0ならばnoargへ
```

```
                lea     arg1,A1         * [illegible]
                bsr     getarg          * [illegible]
                tst.l   D0              * [illegible]
                beq     arg_err         * [illegible]

                lea     arg2,A1         * [illegible]
                bsr     getarg          * [illegible]
                tst.l   D0              * [illegible]
                beq     arg_err         * [illegible]

                lea     arg3,A1         * [illegible]
                bsr     getarg          * [illegible]
                tst.l   D0              * [illegible]
                beq     arg_err         * [illegible]
```

```
        lea     arg3,A0         * arg[3]
        FEFUNC  $20             * __VAL
        bcs     val_err         * エラーならばval_errへ
        PUSH    D0-D1           * arg[3]の倍精度浮動小数点表現

        lea     arg1,A0         * arg[1]
        FEFUNC  $20             * __VAL
        bcs     val_err         * エラーならばval_errへ

        POP     D2-D3           * D0/D1:arg1 , D2/D3:arg3

        move.b  arg2,D4         * arg[2]の先頭文字
        cmp.b   #'+',D4         * '+'?
        beq     plus            *  ならばplusへ
        cmp.b   #'-',D4         * '-'?
        beq     minus           *  ならばminusへ
        cmp.b   #'*',D4         * '*'?
        beq     mul             *  ならばmulへ
        cmp.b   #'/',D4         *'/'?
        beq     div             *  ならばdivへ
        cmp.b   #'Y',D4         *'Y'?
        beq     mod             *  ならばmodへ
        bra     exp_err         *これら以外ならexp_errへ

plus:
        FEFUNC  $2B             * __DADD
        bra     result
minus:
        FEFUNC  $2C             * __DSUB
        bra     result
mul:
        FEFUNC  $2D             * __DMUL
        bra     result
div:
        FEFUNC  $2E             * __DDIV
        bra     result
mod:
        FEFUNC  $2F             * __DMOD
result:
        bcs     calc_err        *  エラーならcalc_errへ

        move.l  #10,D2          * 整数部分の桁数
        move.l  #10,D3          * 少数部分の桁数
        move.l  #%0010000,D4    * アトリビュート('+''-'記号付)
        lea     buffer,A0       * 結果が入るバッファ
        FEFUNC  $21             * __USING

        pea     buffer
        FNC     $09             * print
        addq.l  #4,SP

        pea     crlf_msg
        FNC     $09             * print
        addq.l  #4,SP

        FNC     $00             * exit

noarg:
        pea     noarg_msg
        bra     disp_exit
arg_err:
        pea     arg_err_msg
        bra     disp_exit
val_err:
        pea     val_err_msg
        bra     disp_exit
exp_err:
        pea     exp_err_msg
        bra     disp_exit
calc_err:
        pea     calc_err_msg
disp_exit:
        FNC     $09             * print
        addq.l  #4,SP

        FNC     $00             * exit

getarg:                         * arg取得サブルーチン(簡易型)
        moveq   #0,D0           * 文字数リセット
        moveq   #0,D7           * arg開始フラグリセット
getarg_loop:
        move.b  (A2)+,D1        * 1文字get
        beq     eol             *  $00ならばeolへブランチ
        cmp.b   #' ',D1         * スペース?
        beq     delim           *  ならばdelimへ
        cmp.b   #9,D1           * タブ?
        beq     delim           *  ならばdelimへ
                                * それ以外ならば、
        moveq   #-1,D7          *  arg開始フラグをセット
        move.b  D1,(A1)+        * argバッファにコピー & ポインタ+1
        addq.l  #1,D0           * 文字数+1
        bra     getarg_loop
delim:
        tst.l   D7              * arg開始済み?
        beq     getarg_loop     *  まだならばgetarg_loopへ
eol:
        subq.l  #1,A2           * コマンドラインポインタ補正
        clr.b   (A1)+           * argにエンドマーク

        rts

        .data
noarg_msg:
        dc.b    'usage : fesample <para1> {+|-|*|/|Y} <para2>',13,10
        dc.b    0
arg_err_msg:
        dc.b    '引数が異常です',13,10
        dc.b    0
val_err_msg:
        dc.b    '数値表現が異常です',13,10
        dc.b    0
exp_err_msg:
        dc.b    '演算子が異常です',13,10
        dc.b    0
calc_err_msg:
        dc.b    '計算中にエラーが発生しました',13,10
        dc.b    0
crlf_msg:
        dc.b    13,10,0
        .bss
arg1:
        ds.b    20
arg2:
        ds.b    20
arg3:
        ds.b    20
buffer:
        ds.b    30

        end

*   コマンドラインで指定した式を計算します。項の数は2つで、演算子は'+'
* (加算)、'-'(減算)、'*'(乗算)、'/'(除算)、'Y'(剰余)が使用できます。小
* 数点演算も可能です。項や演算子の間はスペースで区切ってください。
*
* usage:        fesample <para1> {+|-|*|/|Y} <para2>
```

SMPL 1

```
*
*       Function call sample program #01
*               $FF00 exit
*               $FF01 getchar
*               $FF02 putchar
*               $FF07 inkey
*               $FF08 getc
*

                include macro.h

                .text
start:
                FNC     $01             * getchar
*               FNC     $07             * inkey
*               FNC     $08             * getc
                cmp.w   #$1A,D0         * CTRL-Zが入力されたならば
                beq     eof             * プログラムの終了へブランチ
                move.w  D0,result

                bsr     space           * スペース表示サブルーチンコール
                move.w  result,D0
                bsr     hex2            * １６進２桁表示サブルーチンコール
                bsr     space           * スペース表示サブルーチンコール

                bra     start

space:                                  * スペース表示サブルーチン
                move.w  #' ',-(SP)      * スペースを表示
                FNC     $02             * putchar
                addq.l  #2,SP

                rts

eof:
                bsr     crlf            * 改行サブルーチンコール
                FNC     $00             * exit


hex2:                                   * １６進２桁表示サブルーチン
                move.l  D0,-(SP)
                lsr.w   #4,D0
                bsr     hex1
                move.l  (SP)+,D0
hex1:
                and.w   #$0F,D0
                add.w   #'0',D0
                cmp.w   #$3A,D0
                bcs     hex0
                add.w   #$07,D0
hex0:
                move.w  D0,-(SP)
                FNC     $02             * putchar
                addq.l  #2,SP

                rts

crlf:
                move.w  #$0D,-(SP)      * CRコード
                FNC     $02             * putchar
                addq.l  #2,SP
                move.w  #$0A,-(SP)      * LFコード
                FNC     $02             * putchar
                addq.l  #2,SP

                rts
```

```
		.bss
result:
		ds.w	1

		.end
```

```
*   押されたキーのコードを表示するプログラムです。コントロールZを入力
* することでプログラムは終了します。
*   現在、getcharでキー入力を行っていますが、ここをinkey、getcに置き換
* えて、動作の違いを確認してください。
```

SMPL 2

```
*
*	Function call sample program #02
*		$FF03 cominp
*		$FF04 comout
*		$FF06 inpout
*		$FF12 cinsns
*		$FF13 coutsns
*

		include macro.h

		.text
start:
		move.w	#-1,D1
		IOCS	$30		* SET232C (IOCS)
		move.w	D0,D1
		IOCS	$30		* SET232C (IOCS)
					* RS-232Cを初期化する
auxin:
		FNC	$12		* cinsns
		tst.l	D0		* AUXから入力可能でなければ
		beq	keyin		*   keyinにブランチ

		IOCS	$31		* LOF232C (IOCS)
		tst.w	D0		* 受信バッファにデータがなければ
		beq	keyin		*   keyinにブランチ

		FNC	$03		* cominp
		cmp.w	#$FE,D0		* AUXからの入力が$FE以上なら
		bcc	keyin		*   keyinにブランチ (表示の都合)

		move.w	D0,-(SP)
		FNC	$06		* inpout : code≠$FF、≠$FEで画面表示
		addq.l	#2,SP
keyin:
		tst.w	code		* 前のキーの内容が出力できずに残っていたら
		bne	auxout		*   auxoutへブランチ

		move.w	#$FF,-(SP)
		FNC	$06		* inpout : code=$FFでキー入力
		addq.l	#2,SP
		tst.l	D0		* キー入力がなければ
		beq	auxin		*   最初に戻る

		move.w	D0,code		* 押されたキーのコードをワークcodeに保存
auxout:
		FNC	$13		* coutsns
		tst.l	D0		* AUXに出力可能でなければ
		beq	auxin		*   最初に戻る
		move.w	code,-(SP)	* キーボードから入力されたコードを出力
		FNC	$04		* comout
		addq.l	#2,SP
```

```
		clr.w	code		* codeをクリア

		bra	auxin

		.bss
code:
		ds.w	1

		.end
```

* 簡単なターミナルプログラムです。RS-232Cのパラメータ等の設定はSPEED.X
* で行ってください。プログラムを終了するためにはインタラプトボタンを押
* してください。
* コントロールCを押した場合、タイミングによってはcominpのブレークチェッ
* クにひっかかることがあります。

SMPL 3

```
*
*	Function call sample program #03
*		$FF05 prnout
*		$FF11 prnsns
*

		include macro.h

		.text
start:
		tst.b	(A2)+		* コマンドラインの文字数を確認して
		beq	nostr		*   0ならなにもせずに終了

loop1:
		moveq	#0,D0		* D0をクリア
		move.b	(A2)+,D0	* コマンドラインから次の1文字をとってきて
		beq	eol		*   NULLコードなら文字列終了でブランチ

		bsr	out_1char	* プリンタ1文字出力ルーチンコール

		bra	loop1

eol:
		lea	crlf_data,A2	* 改行コードを出力
loop2:
		move.b	(A2)+,D0	* 次の1文字をとってきて
		beq	exit		*   NULLコードならすべ終了

		bsr	out_1char	* プリンタ1文字出力ルーチンコール

		bra	loop2

exit:
nostr:
		FNC	$00		* exit


out_1char:				* プリンタ1文字出力ルーチン
		move.w	D0,code		* コマンドラインの文字をcodeに保存
prn_wait:
		FNC	$11		* prnsns
		tst.l	D0		* PRNへ出力可能？
		beq	prn_wait	*   出力不可ならば、prn_waitへブランチ

		move.w	code,-(SP)
		FNC	$05		* prnout
		addq.l	#2,SP

		rts
```

```
            .data
crlf_data:
            dc.b     13,10,0

            .bss
code:
            ds.w     1

            .end

*   コマンドラインで指定した文字列をプリンタに出力します。文字列には改
* 行コード(CR+LF)を付加します。
*
* usage:        sample03 <strings>
```

SMPL 4

```
*
*       Function call sample program #04
*            $FF09 print
*            $FF0A gets
*            $FF1A getss
*

            include macro.h

            .text
start:
            pea      prompt1          * 入力プロンプトを表示
            FNC      $09              * print
            addq.l   #4,SP

            pea      inpptr           * 文字列を入力
            FNC      $0A              * gets
*           FNC      $1A              * getss
            addq.l   #4,SP

            tst.l    D0               * 入力された文字数をチェック
            beq      null             *   改行のみならば、終了へ

            lea      inpptr+2,A0      * 入力文字列を指すポインタ
            lea      buffer,A1        * 加工後の文字列を納めるﾊﾞｯﾌｧを指すﾎﾟｲﾝﾀ
loop:
            move.b   (A0)+,D0

            cmp.b    #'A',D0
            bcs      notouch
            cmp.b    #'[',D0
            bcs      tosmall          * 大文字ならば小文字化処理へ
            cmp.b    #'a',D0
            bcs      notouch
            cmp.b    #'{',D0
            bcc      notouch
toupper:                              * 小文字ならば大文字化処理
            sub.b    #$20,D0
            bra      notouch
tosmall:
            add.b    #$20,D0
notouch:
            move.b   D0,(A1)+
            bne      loop             * 行端コード(NULL)でなければloopへ

            bsr      crlf2            * 改行サブルーチンコール

            pea      prompt2          * 出力プロンプト表示
```

```
          FNC     $09             * print
          addq.l  #4,SP
          pea     buffer          * 処理結果表示
          FNC     $09             * print
          addq.l  #4,SP

          bsr     crlf2           * 改行サブルーチンコール

          bra     start

null:
          bsr     crlf2           * 改行サブルーチンコール

          FNC     $00             * exit


crlf2:                            * 改行サブルーチン
          pea     crlf2_data      * 改行データ
          FNC     $09             * print
          addq.l  #4,SP

          rts


          .data
prompt1:
          dc.b    '入力：',0
prompt2:
          dc.b    '出力：',0
crlf2_data:
          dc.b    13,10,0
inpptr:
          dc.b    80
          dc.b    0
          ds.b    80+1

          .bss
buffer:
          ds.b    80+1

          .end

*   入力プロンプトに続いて入力した1行の文字列の、大文字は小文字に、小
* 文字は大文字に変換して表示します。漢字には対応していないので正しく表
* 示されないことがあります。プログラムを終了するためにはリターンキーの
* みを押してください。
```

SMPL 5

```
*
*       Function call sample program #05
*               $FF0B keysns
*               $FF0C kflush
*               $FF27 gettim2
*               $FF28 settim2
*

          include macro.h

          text
start:
          move.w  #0,-(SP)        * キーバッファのフラッシュのみ
          FNC     $0C             * kflush
          addq.l  #2,SP

loop:
```

```
        pea     now_msg
        FNC     $09             * print
        addq.l  #4,SP
        FNC     $27             * gettim2
        move.l  D0,D7           * gettim2の返り値をD7に保存
        lsr.l   #8,D0
        lsr.l   #8,D0           * hour
        bsr     putdeci2        * １０進２桁表示ルーチンコール
        bsr     colon           * コロン表示ルーチンコール
        move.l  D7,D0
        lsr.l   #8,D0           * minute
        bsr     putdeci2        * １０進２桁表示ルーチン
        bsr     colon           * コロン表示ルーチンコール
        move.l  D7,D0           * second
        bsr     putdeci2        * １０進２桁表示ルーチン

        pea     change_msg
        FNC     $09             * print
        addq.l  #4,SP

        FNC     $0B             * keysns
        tst.l   D0              * キーは押されている？
        beq     lbl1            *   押されていなければlbl1へ

        FNC     $01             * getchar
        or.b    #$20,D0         * 大文字→小文字処理
        cmp.b   #'y',D0         * 入力文字が'Y'ならば
        beq     inptime         *   inptimeへブランチ
        cmp.b   #'n',D0         * 入力文字が'N'ならば
        beq     nochange        *   nochangeへブランチ
lbl1:
        pea     curmove
        FNC     $09             * print
        addq.l  #4,SP

        bra     loop

inptime:
        pea     prompt
        FNC     $09             * print
        addq.l  #4,SP

        pea     inpptr
        FNC     $0A             * gets
        addq.l  #4,SP
        cmp.w   #8,D0           * 入力文字は８文字？
        bne     inptime         *   でなければ再入力

        moveq   #0,D7
        lea     inpptr+2,A0     * hour
        bsr     getdeci2        * １０進２桁getサブルーチンコール
        tst.l   D0              * エラー？
        bmi     inptime         *   ならば再入力
        or.b    D0,D7
        lsl.l   #8,D7
        lea     inpptr+5,A0     * minute
        bsr     getdeci2        * １０進２桁getサブルーチンコール
        tst.l   D0              * エラー？
        bmi     inptime         *   ならば再入力
        or.b    D0,D7
        lsl.l   #8,D7
        lea     inpptr+8,A0     * second
        bsr     getdeci2        * １０進２桁getサブルーチンコール
        tst.l   D0              * エラー？
        bmi     inptime         *   ならば再入力
        or.b    D0,D7
        move.l  D7,settime      * 設定時刻をsettimeへ

        move.w  #0,-(SP)        * キーバッファのフラッシュのみ
```

```
		FNC	$0C		* kflush
		addq.l	#2,SP

		pea	pushany_msg
		FNC	$09		* print
		addq.l	#4,SP

		move.w	#7,-(SP)	* key flush & inkey
		FNC	$0C		* kflush
		addq.l	#2,SP		* 結果は読み捨てる

		move.l	settime,-(SP)
		FNC	$28		* settim2
		addq.l	#4,SP
		pea	set_msg
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00		* exit

nochange:
		pea	nochange_msg
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00		* exit

putdeci2:					* 10進2桁表示ルーチン
		and.l	#$FF,D0
		divu	#10,D0
		add.l	#$0030_0030,D0
		move.l	D0,D1
		swap	D1
		move.w	D0,-(SP)
		FNC	$02		* putchar
		addq.l	#2,SP
		move.w	D1,-(SP)
		FNC	$02		* putchar
		addq.l	#2,SP
		rts

getdeci2:					* 10進2桁getルーチン
		moveq	#0,D0
		moveq	#0,D1
		move.b	(A0)+,D1
		sub.b	#'0',D1
		bcs	getdeci2_illegal
		cmp.b	#10,D1
		bcc	getdeci2_illegal
		mulu	#10,D1
		move.b	(A0)+,D0
		sub.b	#'0',D0
		bcs	getdeci2_illegal
		cmp.b	#10,D0
		bcc	getdeci2_illegal
		add.w	D1,D0
		rts
getdeci2_illegal:
		moveq	#-1,D0
		rts

colon:
		move.w	#':',-(SP)
		FNC	$02		* putchar
		addq.l	#2,SP
		rts

		.data
```

```
now_msg:
                dc.b    '現在の時刻 : ',0
curmove:
                dc.b    13,27,'[A',0
change_msg:
                dc.b    13,10
                dc.b    '変更しますか？ (Y/N)',27,'[0K',0
prompt:
                dc.b    13,'設定する時刻を入力してください : ',27,'[0K',0
pushany_msg:
                dc.b    13,10
                dc.b    '設定を行う時刻になったら、なにかキーを押してください',0
set_msg:
                dc.b    13,10
                dc.b    '設定しました',13,10
                dc.b    0
nochange_msg:
                dc.b    13,10
                dc.b    '設定は行いませんでした',13,10
                dc.b    0

inpptr:
                dc.b    8
                ds.b    1
                ds.b    8+1

                .bss
settime:
                ds.l    1

                .end

*   時計を設定するためのプログラムです。画面の指示に従って操作を行うこ
* とで、内部時計を正確に設定できます。
* 注）時刻表現は'hh:mm:ss'ですが、コロンでなくてもエラーは発生しません。
*      また、各桁は必ず１０進数２桁で表現してください。
```

SMPL 6

```
*
*       Function call sample program #06
*               $FF0D fflush
*               $FF0F drvctrl
*

                include macro.h

                .text
start:
                tst.b   (A2)+           * コマンドラインの文字数チェック
                beq     nonum           *   ０文字ならなにもせずに終了

                moveq   #0,D0           * D0をクリア
                move.b  (A2),D0         * コマンドラインの最初の１文字
                cmp.b   #'0',D0         * '0' ?
                bcs     num_err         *   より小さければエラー
                cmp.b   #'2',D0         * '2' ?
                bcc     num_err         *   以上ならエラー

                sub.b   #'0'-1,D0
                move.w  D0,drvnum       * ドライブナンバーをdrvnumに保存

                FNC     $0D             * fflush
                                        * 特に必要はないが、念のため
                move.w  #$100,D0        * EJECT
```

```
		or.w	drvnum,D0		* イジェクトするドライブナンバーと合成
		move.w	D0,-(SP)
		FNC	$0F			* drvctrl
		addq.l	#2,SP
nonum:
		FNC	$00			* exit

num_err:
		pea	nonum_msg
		FNC	$09			* print
		addq.l	#4,SP

		FNC	$00			* exit

		.data
nonum_msg:
		dc.b	'ドライブナンバー(0 or 1)を指定してください',13,10
		dc.b	0

		.bss
drvnum:
		ds.w	1

		.end
*	0,1で指定したドライブをイジェクトします。
*
* usage:	sample06 {0,1}
```

SMPL 7

```
*
*	Function call sample program #07
*		$FF0E chgdrv
*		$FF19 curdrv
*

		include macro.h

		.text
start:
		FNC	$19			* curdrv

		addq.w	#1,D0			* 次のドライブ番号
		move.w	D0,nxtdrv		* nextdrvに保存

		move.w	D0,-(SP)
		FNC	$0E			* chgdrv
		addq.l	#2,SP

		cmp.w	nxtdrv,D0		* 指定可能なドライブだった場合
		bgt	noerr			*	noerrへブランチ

		move.w	#0,-(SP)		* ドライブAへ
		FNC	$0E			* chgdrv
		addq.l	#2,SP
noerr:
		FNC	$00			* exit

		.bss
nxtdrv:
		ds.w	1

		.end
*	カレントドライブの次のドライブへ移動します。最後のドライブの次はド
* ライブAへ移動します。
```

SMPL 8

```
*
*       Function call sample program #08
*               $FF17 fatchk
*

                include macro.h

                .text
start:
                tst.b   (A2)+           * コマンドラインの文字数をチェック
                beq     noname          *   文字数が０ならnonameへ

                pea     buffer          * 結果の返るバッファの指定
                move.l  A2,-(SP)        * 調べるファイル名
                FNC     $17             * fatchk
                addq.l  #8,SP

                cmp.l   #8,D0           * セクタは連続している？
                beq     fatcnt          *   連続していればfatcntへ

                pea     msg1
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

fatcnt:
                pea     msg2
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

noname:
                FNC     $00             * exit

                .data
msg1:
                dc.b    'セクタは連続していません。',13,10
                dc.b    0
msg2:
                dc.b    'セクタは連続しています。',13,10
                dc.b    0

                .bss
buffer:
                ds.w    1               * ドライブ番号が入る
                ds.w    2*10            * 先頭セクタ、セクタ数が入る（１０個ぶん）

                .end
*   コマンドラインで指定したファイルが連続したセクタに格納されているか
* どうかチェックします。現在、バッファに返された情報はまったく利用して
* いません。
*
* usage:        sample08 <filename>
```

SMPL 9

```
*
*       Function call sample program #09
*               $FF1B fgetc
*               $FF1D fputc
*               $FF1F allclose
*               $FF3C create
```

```
*               $FF3D open
*               $FF42 seek
*

                include macro.h

                .text
start:
                addq.l  #1,A2

                lea     arg1,A1         * argv[1]
                bsr     getarg          * arg取得サブルーチンコール
                tst.l   D0              * argv[1]文字数
                beq     illegal_cmd     *   0ならillegal_cmdへ

                lea     arg2,A1         * argv[2]
                bsr     getarg          * arg取得サブルーチンコール
                tst.l   D0              * argv[2]文字数
                beq     illegal_cmd     *   0ならillegal_cmdへ

                lea     arg3,A1         * argv[3]
                bsr     getarg          * arg取得サブルーチンコール
                tst.l   D0              * argv[3]文字数
                beq     illegal_cmd     *   0ならillegal_cmdへ

                move.w  #$20,-(SP)      * 通常ファイル
                pea     arg3            * ファイル3
                FNC     $3C             * create
                addq.l  #6,SP
                tst.l   D0              * エラー?
                bmi     create_err      *   ならばcreate_errへ

                move.w  D0,handl3       * ファイルハンドルをhandl3に保存

                move.w  #0,-(SP)        * READ
                pea     arg1            * ファイル1
                FNC     $3D             * open
                addq.l  #6,SP
                tst.l   D0              * エラー?
                bmi     open1_err       *   ならばopen1_errへ

                move.w  D0,handl1       * ファイルハンドルをhandl1に保存

                move.w  #0,-(SP)        * READ
                pea     arg2            * ファイル2
                FNC     $3D             * open
                addq.l  #6,SP
                tst.l   D0              * エラー?
                bmi     open2_err       *   ならばopen2_errへ

                move.w  D0,handl2       * ファイルハンドルをhandl2に保存
                clr.l   D7              * ファイル2のオフセットクリア
loop:
                move.w  handl1,-(SP)    * ファイル1から1文字入力
                FNC     $1B             * fgetc
                addq.l  #2,SP
                tst.l   D0              * ファイル1終端?
                bmi     eof

                move.l  D0,D1
lbl1:
                move.w  handl2,-(SP)    * ファイル2から1文字入力
                FNC     $1B             * fgetc
                addq.l  #2,SP
                tst.l   D0              * ファイル2終端?
                bpl     lbl2            *   でなければlbl2へ

                tst.l   D7              * ファイル2の文字数が0なら
```

```
        beq     too_short_err   *   too_short_errへ

        move.w  #0,-(SP)        * ファイル先頭から
        move.l  #0,-(SP)        * オフセット0に移動(つまりファイル先頭)
        move.w  handl2,-(SP)    * ファイル2
        FNC     $42             * seek
        addq.l  #8,SP
        clr.l   D7              * ファイル2のオフセットクリア

        bra     lbl1
lbl2:
        addq.l  #1,D7           * ファイル2オフセット+1
        eor.b   D0,D1           * ファイル1と2の文字をXOR

        move.w  handl3,-(SP)    * ファイル3
        move.w  D1,-(SP)        * XORの結果
        FNC     $1D             * fputc
        addq.l  #4,SP

        bra     loop
eof:
        FNC     $1F             * allclose

        FNC     $00             * exit

illegal_cmd:
        pea     illegal_cmd_msg
        bra     disp_abort
open1_err:
        pea     open1_err_msg
        bra     disp_abort
open2_err:
        pea     open2_err_msg
        bra     disp_abort
create_err:
        pea     create_err_msg
        bra     disp_abort
too_short_err:
        pea     too_short_err_msg
disp_abort:
        FNC     $09             * print
        addq.l  #4,SP

        FNC     $00             * exit


getarg:                         * arg取得サブルーチン(簡易型)
        moveq   #0,D0           * 文字数リセット
        moveq   #0,D7           * arg開始フラグリセット
getarg_loop:
        move.b  (A2)+,D1        * 1文字get
        beq     eol             *   $00ならばeolへブランチ
        cmp.b   #' ',D1         * スペース?
        beq     delim           *   ならばdelimへ
        cmp.b   #9,D1           * タブ?
        beq     delim           *   ならばdelimへ
                                * それ以外ならば..
        moveq   #-1,D7          *   arg開始フラグをセット
        move.b  D1,(A1)+        * argバッファにコピー & ポインタ+1
        addq.l  #1,D0           * 文字数+1
        bra     getarg_loop
delim:
        tst.l   D7              * arg開始済み?
        beq     getarg_loop     *   まだならばgetarg_loopへ
eol:
        subq.l  #1,A2           * コマンドラインポインタ補正
        clr.b   (A1)+           * argにエンドマーク

        rts
```

```
                .data
illegal_cmd_msg:
                dc.b    'パラメータの指定が間違っています',13,10
                dc.b    0
open1_err_msg:
                dc.b    'ファイル1をオープンできません',13,10
                dc.b    0
open2_err_msg:
                dc.b    'ファイル2をオープンできません',13,10
                dc.b    0
create_err_msg:
                dc.b    'ファイル3を作成できません',13,10
                dc.b    0
too_short_err_msg:
                dc.b    'ファイル2の内容がありません',13,10
                dc.b    0

                .bss
arg1:
                ds.b    40
arg2:
                ds.b    40
arg3:
                ds.b    40

handl1:
                ds.w    1
handl2:
                ds.w    1
handl3:
                ds.w    1

                .end

*   コマンドラインで指定した3つのファイル、ファイル1、ファイル2、ファ
* イル3について、ファイル2の内容をキーとしてファイル1の内容を暗号化
* し、その結果をファイル3に書き込みます。
*   同じキーを使用して、ファイル3として作成したファイルに暗号処理を施
* すことでもとのファイル1と同じ内容を復元することができます。
*
* usage:        sample09 <file1> <file2> <file3>
```

SMPL 10

```
*
*       Function call sample program #10
*               $FF1C fgets
*               $FF1E fputs
*

                include macro.h

                .text
start:
                tst.b   (A2)+           * コマンドラインの文字数のチェック
                beq     noname          *   0ならばnonameへ

                move.w  #$00,-(SP)      * READ
                move.l  A2,-(SP)        * コマンドラインがファイルネーム
                FNC     $3D             * open
                addq.l  #6,SP

                tst.l   D0              * エラーなら
                bmi     openerr         *   openerrへブランチ

                move.w  D0,handl        * ファイルハンドルをhandlに保存
```

```
        move.w  D0,-(SP)
        pea     inpptr1
        FNC     $1C             * fgets
        addq.l  #6,SP

        move.w  #1,-(SP)        * 標準出力へ出力
        pea     headmsg
        FNC     $1E             * fputs
        addq.l  #6,SP

        move.w  #1,-(SP)        * 標準出力へ出力
        pea     inpptr1+2
        FNC     $1E             * fputs
        addq.l  #6,SP

        bsr     crlf

        clr.b   inpptr1+2       * バッファ1をクリア
        clr.b   inpptr2+2       * バッファ2をクリア
        moveq   #0,D7           * D7は入力バッファの切り替えスイッチ
loop:
        not.w   D7              * スイッチを反転
        bne     inp2
inp1:
        lea     inpptr1,A1      * スイッチが0になったらinpptr1で入力
        lea     inpptr2,A2      * inpptr2で表示
        bra     inp
inp2:
        lea     inpptr2,A1      * スイッチが-1になったらinpptr2で入力
        lea     inpptr1,A2      * inpptr1でで表示
inp:
        move.w  handl,-(SP)
        move.l  A1,-(SP)
        FNC     $1C             * fgets
        addq.l  #6,SP

        tst.l   D0              * 入力文字数が0(ファイル終端)か?
        bpl     loop            *   そうでなければloopへ

        move.w  #1,-(SP)        * 標準出力へ出力
        pea     tailmsg
        FNC     $1E             * fputs
        addq.l  #6,SP

        addq.l  #1,A2
        move.w  #0,-(SP)        * 標準出力へ出力
        move.l  A2,-(SP)
        FNC     $1E             * fputs
        addq.l  #6,SP

        bsr     crlf

        move.w  handl,-(SP)
        FNC     $3E             * close
        addq.l  #2,SP
noname:
        FNC     $00             * exit

openerr:
        move.w  #2,-(SP)        * 標準エラー出力へ出力
        pea     openerr_msg
        FNC     $1E             * fputs
        addq.l  #6,SP

        FNC     $00

crlf:                           * 改行サブルーチン
        move.w  #1,-(SP)        * 標準出力へ出力
        pea     crlf_msg
```

```
		FNC	$1E		* fputs
		addq.l	#6,SP

		rts

		.data
headmsg:
		dc.b	'HEAD:',0
tailmsg:
		dc.b	'TAIL:',0
openerr_msg:
		dc.b	'ファイルがオープンできませんでした',13,10
		dc.b	0
crlf_msg:
		dc.b	13,10,0

inpptr1:
		dc.b	80		* 最大８０文字
		ds.b	1
		ds.b	80+1
inpptr2:
		dc.b	80		* 最大８０文字
		ds.b	1
		ds.b	80+1

		.bss
		.even
handl:
		ds.w	1

		.end

*	コマンドラインで指定したファイルの先頭の行と最終行を表示します。
*	プログラム中では表示を行うために標準出力ハンドルへのfputsを行ってい
* ますが、もちろん＄ＦＦ０９printを使用しても構いません。ただし、エラー
* の表示にだけは標準エラー出力を利用しています。そのため、リダイレクト
* を指定してもエラー表示だけはリダイレクトされないことを確認してくださ
* い。
** usage:   sample10 <filename>
```

SMPL 11

```
*
*       Function call sample program #11
*               $FF20 super
*

                include macro.h

                .text
start:
                clr.l   -(SP)           * スーパーバイザモードへ
                FNC     $20             * super
                addq.l  #4,SP
                move.l  D0,_usp         * USPの内容を_uspに保存

                move.b  $e8e007,D0      * スーパーバイザエリアにアクセスできる
                eor.b   #%0001,D0       * H/L Res.LED 反転
                move.b  D0,$e8e007

                move.l  _usp,-(SP)      * ユーザーモードへ復帰
                FNC     $20             * super
                addq.l  #4,SP

                FNC     $00             * exit

                .bss
_usp:
                ds.l    1

                .end

*   特権状態をスーパーバイザモードに切り替え、直接I／Oポートにアクセ
* スしてみます。このプログラムではH/L Res.LEDを反転します。HDタイプの
* X68Kでは結果を確認できません。
```

SMPL 12

```
*
*       Function call sample program #12
*               $FF21 fnckey
*               $FF23 conctrl
*

                include macro.h

FEFUNC          macro   number
                dc.b    $FE
                dc.b    number
                endm

                .text
start:
                tst.b   (A2)+           * コマンドラインの文字数チェック
                beq     noarg           *   ならばnoargへ

                lea     arg1,A1         * argv[1]
                bsr     getarg          * arg取得サブルーチンコール
                tst.l   D0              * 文字数0?
                beq     arg_err         *   ならばarg_errへ

                lea     arg2,A1         * argv[2]
                bsr     getarg          * arg取得サブルーチンコール
```

```
		tst.l	D0		* 文字数０？
		beq	arg_err		*   ならばarg_errへ

		lea	arg1,A0		* argv[1]を数値に変換
		FEFUNC	$10		* __STOL (FEFUNC)
		bcs	keynum_err	*   エラーならばkeynum_errへ
		cmp.b	#1,D0		* １より小さい？
		bcs	keynum_err	*   ならばkeynum_err
		cmp.b	#21,D0		* ２１以上？
		bcc	keynum_err	*   ならばkeynum_err

		pea	arg2		* argv[2]をＦＰキーに設定
		add.w	#$100,D0	* 設定モード
		move.w	D0,-(SP)
		FNC	$21		* keyctrl
		addq.l	#6,SP

		move.w	#-1,-(SP)	* 現在のＦＰキー行のモードを得る
		move.w	#14,-(SP)
		FNC	$23		* conctrl
		addq.l	#4,SP
		move.w	D0,orgmode	* orgmodeに保存

		move.w	#5,D1
loop:
		move.w	#0,-(SP)	* ＦＰキー行表示
		move.w	#14,-(SP)
		FNC	$23		* conctrl
		addq.l	#4,SP

		bsr	wait		* waitサブルーチンコール

		move.w	#2,-(SP)	* ＦＰキー行非表示
		move.w	#14,-(SP)
		FNC	$23		* conctrl
		addq.l	#4,SP

		bsr	wait		* waitサブルーチンコール

		dbra	D1,loop
blink_end:
		move.w	orgmode,-(SP)	* ＦＰキー行再設定
		move.w	#14,-(SP)
		FNC	$23		* conctrl
		addq.l	#4,SP

		FNC	$00		* exit

noarg:
		pea	noarg_msg
		bra	disp_abort
arg_err:
		pea	arg_err_msg
		bra	disp_abort
keynum_err:
		pea	keynum_err_msg
disp_abort:
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00

getarg:					* arg取得サブルーチン
		moveq	#0,D0		* 文字数リセット
		moveq	#0,D6		* quoteフラグリセット
		moveq	#0,D7		* arg開始フラグリセット
getarg_loop:
		move.b	(A2)+,D1	* １文字get
```

```
		beq	eol		*	$00ならばeolへブランチ
		cmp.b	#$22,D1		*	ダブルクォート?
		beq	quote		*	ならばquoteへ
		cmp.b	#' ',D1		*	スペース?
		beq	delim		*	ならばdelimへ
		cmp.b	#9,D1		*	タブ?
		beq	delim		*	ならばdelimへ
other:					*	それ以外ならば..
		moveq	#-1,D7		*	arg開始フラグをセット
		move.b	D1,(A1)+	*	argバッファにコピー & ポインタ+1
		addq.l	#1,D0		*	文字数+1
		bra	getarg_loop
quote:
		tst.l	D6		*	quote開始済み?
		bne	eol		*	ならばeolへ
		moveq	#-1,D6
		bra	getarg_loop
delim:
		tst.l	D7		*	arg開始済み?
		beq	getarg_loop	*	まだならばgetarg_loopへ
		tst.l	D6		*	quote開始済み?
		bne	other		*	ならばotherへ
eol:
		subq.l	#1,A2		*	コマンドラインポインタ補正
		clr.b	(A1)+		*	argにエンドマーク

		rts

wait:					*	ウェイトサブルーチン
		move.w	#16384,D2
		dbra	D2,*-2		*	その場でwait
wait_end:
		rts

		.data
noarg_msg:
		dc.b	'usage : sample12 <key#> <str>',13,10
		dc.b	0
arg_err_msg:
		dc.b	'パラメータが異常です',13,10
		dc.b	0
keynum_err_msg:
		dc.b	'キー番号が異常です',13,10
		dc.b	0

		.bss
arg1:
		ds.b	40
arg2:
		ds.b	40
orgmode:
		ds.w	1

		.end

*	コマンドラインの第1パラメータで指定したファンクションキーに、第2
*	パラメータの文字列を設定します。KEY.Xでも同様な操作が可能ですが、この
*	プログラムの場合、文字列をダブルクォーテーションで囲むことでデリミタ
*	(スペースやタブ)もキーに設定できます。
*	設定後、FPキー表示部をブリンクさせて注意を引きます。
*
*	usage:		sample12 <key#> <string>
```

SMPL 13

```
*
*       Function call sample program #13
*               $FF24 keyctrl
*

                include macro.h

                .text
start:
                move.w  #2,-(SP)        * シフトキー類の状態を得る
                FNC     $24             * keyctrl
                addq.l  #2,SP

                and.w   #$0003,D0
                move.w  D0,-(SP)        * エラーコード
                FNC     $4C             * exit2

                .end

*   ＣＴＲＬキーとＳＨＩＦＴキーの状態をエラーコードとして返すプログラ
* ムです。
*   ０：ＳＨＩＦＴもＣＴＲＬも押されていない
*   １：ＳＨＩＦＴが押されている
*   ２：ＣＴＲＬが押されている
*   ３：ＳＨＩＦＴとＣＴＲＬが押されている
* 以下のようにバッチファイルで使用するとよいでしょう。
* --------------------------------------------------------------------
*           :
*  sample13
*  if exitcode 1 goto SHIFT
*  if exitcode 2 goto CTRL
*           :
* :SHIFT
*  echo "SHIFTが押されています"
*           :
* --------------------------------------------------------------------
```

SMPL 14

```
*
*       Function call sample program #14
*               $FF25 intvcs
*               $FF35 intvcg
*               $FF31 keeppr
*

                include macro.h

                .text
TOP:            equ     *

process_name:
                dc.b    'SAMPLE14'
original_adr2:
                ds.l    1               * もともとのルーチンへのアドレス
main:                                   * 割り込みで駆動される部分
                PUSH    D0-D2/A1

                lea     pcmdat,A1
                move.w  #$0403,D1       * 15.6kHz,R/L
                move.l  datalen,D2
                IOCS    $60             * ADPCMOUT (IOCS)

                POP     D0-D2/A1
```

```
		dc.w	$4EF9		* = jmp
original_adr:
		ds.l	1		* もともとのルーチンのアドレスへジャンプ

datalen:
		ds.l	1

start:					* 初期化部分のスタート
		clr.l	-(SP)		* スーパーバイザモードへ
		FNC	$20		* super
		addq.l	#4,SP		* 返り値は読み捨てる
					* （ユーザモードへの復帰を省略したため）
		move.w	#$61,-(SP)	* INTベクター#$61 : 2HD挿入／イジェクト割り
込み
		FNC	$35		* intvcg
		addq.l	#2,SP

		move.l	D0,A1
		move.l	A1,original_adr * オリジナルのベクタを保存
		move.l	A1,original_adr2

		lea	-12(A1),A1	* process_name
		cmp.l	#'SAMP',(A1)	* process_nameの前4バイトは一致するか？
		bne	not_exist	*   一致しなければnot_existへ
		cmp.l	#'LE14',4(A1)	* process_nameの後ろ4バイトは一致するか？
		bne	not_exist	*   一致しなければnot_existへ
exist:					* すでにSAMPLE14が組み込まれていた場合
		move.l	8(A1),-(SP)	* もともとのルーチンのアドレス
		sub.l	#$F0,A1		* PSP+$10を求める
		move.w	#$61,-(SP)	* ベクター番号
		FNC	$25		* intvcs
		addq.l	#6,SP

		move.l	A1,-(SP)	* 常駐していたSAMPLE14のPSP+$10
		FNC	$49		* mfree
		addq.l	#4,SP

		pea	free_msg
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00		* exit

not_exist:
		tst.b	(A2)+		* コマンドラインの文字数をチェック
		beq	noname		*   0ならばnonameへ

		move.w	#0,-(SP)	* READ
		move.l	A2,-(SP)
		FNC	$3D		* open
		addq.l	#6,SP
		tst.l	D0		* エラー？
		bmi	open_err	*   ならばopen_errへ

		move.l	D0,D7		* ファイルハンドルをD7に待避

		move.l	#$FE00,-(SP)	* $FE00バイト以内をロード
		pea	pcmdat
		move.w	D7,-(SP)
		FNC	$3F		* read
		lea	10(SP),SP
		tst.l	D0		* エラー？
		bmi	read_err	*   ならばread_errへ

		move.l	D0,datalen	* 実際に読み込んだバイト数をdatalenへ保存

		move.w	D7,-(SP)
		FNC	$3E		* close
		addq.l	#2,SP
```

```
		pea	main		* 新しいルーチンのアドレス
		move.w	#$61,-(SP)	* ベクター番号
		FNC	$25		* intvcs
		addq.l	#6,SP
		pea	keep_msg
		FNC	$09		* print
		addq.l	#4,SP

		lea	pcmdat-TOP,A0
		add.l	datalen,A0
		move.w	#0,-(SP)	* 終了コード0
		move.l	A0,-(SP)	* 常駐させるバイト数
		FNC	$31		* keeppr

noname:
		pea	noname_msg
		bra	disp_abort
open_err:
		pea	open_err_msg
		bra	disp_abort
read_err:
		pea	read_err_msg
disp_abort:
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00		* exit

noname_msg:
		dc.b	'ファイルネームを指定してください',13,10
		dc.b	0
open_err_msg:
		dc.b	'ファイルをオープンできませんでした',13,10
		dc.b	0
read_err_msg:
		dc.b	'ファイル読み込みでエラーが発生しました',13,10
		dc.b	0
free_msg:
		dc.b	'常駐解除しました',13,10
		dc.b	0
keep_msg:
		dc.b	'常駐します',13,10
		dc.b	0

pcmdat:

		.end	start		* startから実行を開始することを指示

*	ベクター$61（2HDディスク挿入／イジェクト割り込み）を書き換え
* て、ディスクの挿入／イジェクト時にコマンドラインで指定したファイルの
* 内容をPCMデータとして発音します。
*	一度実行すると、それ以降挿入／イジェクトのたびにPCMデータが発音
* されます。もう一度実行すると、プログラムの常駐を解除し、占有していた
* メモリを開放します。
*	PCMデータファイルには、15kHzでサンプリングしたデータを納めておい
* てください。
*
* usage:	sample14 [<file_name>]
*			↑常駐解除時には不用
```

SMPL 15

```
*
*       Function call sample program #15
*               $FF26 pspset
*               $FF48 malloc
*               $FF49 mfree
*               $FF4A setblock
*               $FF50 setpdb
*               $FF51 getpdb
*

                include macro.h

                .text
start:
                move.l  #BOTTOM-start+$F0,-(SP) * 自分に必要最小限のメモリ
                pea     16(A0)          * 現在のPSP
                FNC     $4A             * setblock
                addq.l  #8,SP

                FNC     $51             * getpdb
                move.l  D0,nowpdb       * 現在のPDBADRをnowpdbに保存

                move.l  #$200,-(SP)     * $200バイト確保
                FNC     $48             * malloc
                addq.l  #4,SP
                tst.l   D0              * エラー?
                bmi     alloc_err       *   ならばalloc_errへ

                move.l  D0,memptr       * 確保したメモリのポインタをmemptrへ

                move.l  D0,-(SP)
                FNC     $26             * pspset
                addq.l  #4,SP   * ↓ここから下は新しいﾌﾟﾛｾｽとして実行されています
                                * -------------------------------------------------
--
                move.l  memptr,A0       * +
                lea     -$10(A0),A0     * +
                move.w  #'?:',$80(A0)   * + 新しいプロセスのドライブ名(1)
                clr.b   $82(A0)         * + 新しいプロセスのドライブ名(2)
                move.l  #'子ブ',$C4(A0) * + 新しいプロセスの名前(1)
                move.l  #'ロ1',$C8(A0)  * + 新しいプロセスの名前(2)
                clr.b   $CC(A0)         * + 新しいプロセスの名前(3)

                clr.l   -(SP)           * + 現在の環境
                pea     P1              * + コマンドライン: /c process
                pea     FILE            * + ファイル名     : a:¥command.x
                move.w  #0,-(SP)        * + load & exec
                FNC     $4B             * + exec
                lea     14(SP),SP       * +
                move.l  D0,-(SP)        * + execの返り値をプッシュ

                move.l  nowpdb,-(SP)    * +
                FNC     $50             * setpdb ------------------------------------
--
                addq.l  #4,SP           * ↑これで元のプロセスに復帰しました

                move.l  (SP)+,D0        * execの返り値をポップ

                tst.l   D0              * エラー?
                bmi     exec_err        *   ならばexec_errへ

                move.l  memptr,-(SP)    * 新しいプロセスのメモリを開放
                FNC     $49             * mfree
                addq.l  #4,SP

                FNC     $00             * exit
```

```
alloc_err:
                pea     alloc_err_msg
                bra     disp_abort
exec_err:
                pea     exec_err_msg
disp_abort:
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

FILE:
                dc.b    'a:¥command.x',0
P1:
                dc.b    10
                dc.b    '/c process',0

alloc_err_msg:
                dc.b    'メモリを確保できません',13,10
                dc.b    0
exec_err_msg:
                dc.b    'command.xを実行できません',13,10
                dc.b    0

                .even
nowpdb:
                ds.l    1
memptr:
                ds.l    1

BOTTOM:         equ     *

                .end

*   新しいプロセスをメモリ上に作成してみます。表示されるプロセスの状況
* の中で"子プロ1"と表示されているのが作成したプロセスです。
*   プロセスの状態を表示するために"process.x"を子プロセスとして起動して
* います。そのため、このプログラムを実行する場合には、以下のような条件
* が必要です。
* ★"command.x"がドライブAのルートディレクトリに存在する。
* ★"process.x"が実行パスに存在する。
```

SMPL 16

```
*
*       Function call sample program #16
*               $FF29 namests
*

                include macro.h

                .text
start:
                tst.b   (A2)+           * コマンドラインの文字数をチェック
                beq     noname          *  0ならばnonameへ

                pea     buffer
                move.l  A2,-(SP)
                FNC     $29             * namests
                addq.l  #8,SP

                lea     buffer,A1

                pea     drvmsg
                FNC     $09             * print
                addq.l  #4,SP
                moveq   #0,D0
```

```
        move.b  1(A1),D0        * ドライブ番号
        bsr     hex2
        bsr     crlf

        pea     pathmsg
        FNC     $09             * print
        addq.l  #4,SP
        pea     2(A1)           * パス名
        FNC     $09             * print
        addq.l  #4,SP
        bsr     crlf

        move.b  75(A1),D7       * 拡張子の最初の文字をD7に保存
        clr.b   75(A1)          * NULLコード追加の都合
        pea     name1msg
        FNC     $09             * print
        addq.l  #4,SP
        pea     67(A1)          * ファイルネーム（8文字目まで）
        FNC     $09
        addq.l  #4,SP
        bsr     crlf

        move.b  D7,75(A1)       * 拡張子の最初の文字を復帰
        move.b  78(A1),D7       * ファイル名残りの最初の文字をD7に保存
        clr.b   78(A1)          * NULLコード追加の都合
        pea     extmsg
        FNC     $09             * print
        addq.l  #4,SP
        pea     75(A1)          * 拡張子
        FNC     $09
        addq.l  #4,SP
        bsr     crlf

        move.b  D7,78(A1)       * ファイル名残りの最初の文字を復帰
        pea     name2msg
        FNC     $09             * print
        addq.l  #4,SP
        pea     78(A1)          * ファイルネーム（8文字目まで）
        FNC     $09
        addq.l  #4,SP
        bsr     crlf
noname:
        FNC     $00             * exit

hex2:                           * 16進2桁表示ルーチン
        move.l  D0,-(SP)
        lsr.w   #4,D0
        bsr     hex1
        move.l  (SP)+,D0
hex1:
        and.w   #$0F,D0
        add.w   #'0',D0
        cmp.w   #$3A,D0
        bcs     hex0
        add.w   #7,D0
hex0:
        move.w  D0,-(SP)
        FNC     $02             * putchar
        addq.l  #2,SP

        rts

crlf:                           * 改行ルーチン
        pea     crlf_msg
        FNC     $09             * print
        addq.l  #4,SP
        rts
```

```
		.data
drvmsg:
		dc.b	'ドライブ番号	:$',0
pathmsg:
		dc.b	'パス名		:',0
name1msg:
		dc.b	'ファイル名	:',0
extmsg:
		dc.b	'拡張子		:',0
name2msg:
		dc.b	'ファイル名(残り):',0
crlf_msg:
		dc.b	13,10,0

		.bss
		.even
buffer:
		ds.b	88

		.end

*	コマンドラインで指定したファイルについて、namestsで得られる情報を表
* 示します。
*
* usage:	sample16 <filename>
```

SMPL 17

```
*
*	Function call sample program #17
*		$FF2A getdate
*		$FF2B setdate
*		$FF2C gettime
*		$FF2D settime
*		$FF57 filedate
*

		include macro.h

		.text
start:
		tst.b	(A2)+		* コマンドラインの文字数チェック
		beq	noname		*	0ならばnonameへ

		FNC	$2A		* getdate
		move.w	D0,cur_date	* 現在時刻をcur_dateに待避
		FNC	$2C		* gettime
		move.w	D0,cur_time	* 現在時刻をcur_timeに待避

		move.w	#2,-(SP)	* READ/WRITE
		move.l	A2,-(SP)
		FNC	$3D		* open
		addq.l	#6,SP
		tst.l	D0		* エラー?
		bmi	open_err	*	ならばopen_errへ

		move.w	D0,D7		* ファイルハンドルをD7に待避

		move.l	#0,-(SP)	* ファイルの日付を調べるだけ
		move.w	D7,-(SP)
		FNC	$57		* filedate
		addq.l	#6,SP
		move.l	D0,file_date_time
		swap.w	D0		* filedateのエラーは上位ワードで判定
		cmp.w	#$FFFF,D0	* エラー?
		beq	date_err	*	ならばdate_errへ
```

```
                move.w  cur_date,D0
                swap.w  D0
                move.w  cur_time,D0
                move.l  D0,-(SP)        * 現在の日付／時刻を設定
                move.w  D7,-(SP)
                FNC     $57             * filedate
                addq.l  #6,SP

                move.l  file_date_time,D1
                move.w  D1,-(SP)        * ファイルの日付を現在の日付／時刻とする
                FNC     $2D             * settime
                addq.l  #2,SP
                swap.w  D1
                move.w  D1,-(SP)
                FNC     $2B             * setdate
                addq.l  #2,SP

                move.w  D7,-(SP)
                FNC     $3E             * close
                addq.l  #2,SP

                FNC     $00             * exit

noname:
                pea     noname_msg
                bra     disp_abort
open_err:
                pea     open_err_msg
                bra     disp_abort
date_err:
                pea     date_err_msg
disp_abort:
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

                .data
noname_msg:
                dc.b    'ファイル名を指定してください',13,10
                dc.b    0
open_err_msg:
                dc.b    'ファイルがみつかりません',13,10
                dc.b    0
date_err_msg:
                dc.b    'ファイルの日付を読み出せません',13,10
                dc.b    0

                .bss
cur_date:
                ds.w    1
cur_time:
                ds.w    1
file_date_time:
                ds.l    1

                .end

*   コマンドラインで指定したファイルの日付／時刻と現在の日付／時刻を入
* れ替えます。
*   実用的な意味はほとんどありません。実行後は内蔵時計の日付と時刻を修
* 正することをお勧めします。
*
* usage:        sample17 <filename>
```

SMPL 18

```
*
*       Function call sample program #18
*               $FF30 vernum
*

                include macro.h

                .text
start:
                FNC     $30             * vernum

                move.l  D0,-(SP)

                pea     osname
                FNC     $09             * print
                addq.l  #4,SP

                move.l  (SP)+,D0

                bsr     hex4@

                pea     crlf_msg
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

hex4@:                                  * １６進４桁表示サブルーチン(小数点付)
                move.l  D0,-(SP)
                lsr.w   #8,D0
                bsr     hex2

                move.w  #'.',-(SP)
                FNC     $02             * putchar
                addq.l  #2,SP

                move.l  (SP)+,D0
hex2:
                move.l  D0,-(SP)
                lsr.w   #4,D0
                bsr     hex1
                move.l  (SP)+,D0
hex1:
                and.w   #$0F,D0
                add.w   #'0',D0
                cmp.w   #$3A,D0
                bcs     hex0
                add.w   #7,D0
hex0:
                move.w  D0,-(SP)
                FNC     $02             * putchar
                addq.l  #2,SP

                rts

                .data
osname:
                dc.b    'Human68k version ',0
crlf_msg:
                dc.b    13,10,0

                .end
```

*　Ｈｕｍａｎ６８ｋのバージョンを表示します。

SMPL 19

```
;
;       Function call sample program #19
;               $FF32 getdpb
;

                include macro.h

                .text
ent:
                pea     dpbptr
                move.w  #0,-(SP)                * カレントドライブ
                FNC     $32                     * getdpb
                addq.l  #6,SP

                pea     msg1
                FNC     $09                     * print
                addq.l  #4,SP

                move.l  dpbptr+18,D0            * 装置ドライバへのポインタ
                bsr     hex8                    * １６進８桁表示ルーチンコール

                pea     msg2
                FNC     $09                     * print
                addq.l  #4,SP

                FNC     $00                     * exit

hex8:                                           * １６進８桁表示サブルーチン
                move.l  D0,-(SP)
                swap    D0
                bsr     hex4
                move.l  (SP)+,D0
hex4:
                move.l  D0,-(SP)
                lsr.w   #8,D0
                bsr     hex2
                move.l  (SP)+,D0
hex2:
                move.l  D0,-(SP)
                lsr.w   #4,D0
                bsr     hex1
                move.l  (SP)+,D0
hex1:
                and.w   #$0F,D0
                add.w   #'0',D0
                cmp.w   #$3A,D0
                bcs     hex0
                add.w   #7,D0
hex0:
                move.w  D0,-(SP)
                FNC     $02                     * putchar
                addq.l  #2,SP

                rts

                .data
msg1:
                dc.b    'カレントドライブのデバイスドライバ : $',0
msg2:
                dc.b    13,10,0

                .bss
dpbptr:
                ds.b    94

                .end
```

SMPL 20

```
*
*       Function call sample program #20
*               $FF33 breakck
*

                include macro.h

                .text
start:
                move.w  #-1,-(SP)       * BREAKフラグの設定状況を調べる
                FNC     $33             * breakck
                addq.l  #2,SP

                tst.l   D0              * BREAK = OFF ?
                beq     turn_on         *   ならばturn_onへ
                cmp.l   #1,D0           * BREAK = ON ?
                beq     turn_off        *   ならばturn_offへ

                pea     break_kill_msg  * それ以外ならBREAK = KILL(V2.00)と判断
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

turn_on:
                move.w  #1,D0           * BREAK = ON
                lea     break_on_msg,A1
                bra     set_break
turn_off:
                move.w  #0,D0           * BREAK = OFF
                lea     break_off_msg,A1
set_break:
                move.w  D0,-(SP)
                FNC     $33             * breakck
                addq.l  #4,SP

                move.l  A1,-(SP)
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

                .data
break_on_msg:
                dc.b    'BREAK を <on> にしました',13,10
                dc.b    0
break_off_msg:
                dc.b    'BREAK を <off> にしました',13,10
                dc.b    0
break_kill_msg:
                dc.b    'BREAK は <kill> 状態です',13,10
                dc.b    0

                .end

*   ブレークフラグをチェックし、<on>、<off>を切り替えます。Human2.00で
* <kill>を指定している場合はなにもしません。
```

SMPL 21

```
*
*       Function call sample program #21
*               $FF34 drvxchg
*
```

```
        include macro.h

        .text
start:
        tst.b   (A2)+           * コマンドラインの文字数をチェック
        beq     noname          *   0ならばnonameへ

        moveq.l #0,D0
        move.b  (A2),D0         * 1文字取ってくる
        or.b    #$20,D0         * 大文字対応処理
        sub.b   #$60,D0         * ドライブ番号

        move.w  #0,-(SP)        * カレントドライブ
        move.w  D0,-(SP)
        FNC     $34             * drvxchg
        addq.l  #4,SP

        FNC     $00             * exit

noname:
        pea     msg1
        FNC     $09             * print
        addq.l  #4,SP

        FNC     $00             * exit

        .data
msg1:
        dc.b    'ドライブ名を指定してください',13,10
        dc.b    0

        .end

*   コマンドラインで指定したドライブとカレントドライブを入れ替えます。
*
* usage:        sample21 <drive_name>
```

SMPL 22

```
*
*       Function call sample program #22
*               $FF36 dskfre
*

        include macro.h

        .text
start:
        tst.b   (A2)+           * コマンドラインの文字数のチェック
        beq     noname          *   文字数0ならnonameへ

        moveq   #0,D1
        move.b  (A2),D1
        or.b    #$20,D1         * 小文字処理
        sub.b   #$60,D1         * D1にはドライブ番号

        pea     buffer
        move.w  D1,-(SP)
        FNC     $36             * dskfre
        addq.l  #6,SP
        tst.l   D0              * エラー?
        bmi     error           *   ならばerrorへ

        pea     msg1
        FNC     $09             * print
        addq.l  #4,SP
```

```
                lea     buffer,A1
                move.w  4(A1),D0        * １クラスタあたりのセクタ数
                mulu    6(A1),D0        * １セクタあたりのバイト数
                mulu    (A1),D0         * 使用可能なクラスタ数

                bsr     hex8            * １６進８桁表示サブルーチンコール

                pea     crlf_msg
                FNC     $09             * print
                addq.l  #4,SP
noname:
                FNC     $00             * exit

error:
                pea     err_msg
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00

hex8:                                   * １６進８桁表示サブルーチン
                move.l  D0,-(SP)
                swap    D0
                bsr     hex4
                move.l  (SP)+,D0
hex4:
                move.l  D0,-(SP)
                lsr.w   #8,D0
                bsr     hex2
                move.l  (SP)+,D0
hex2:
                move.l  D0,-(SP)
                lsr.w   #4,D0
                bsr     hex1
                move.l  (SP)+,D0
hex1:
                and.w   #$0F,D0
                add.w   #'0',D0
                cmp.w   #$3A,D0
                bcs     hex0
                add.w   #7,D0
hex0:
                move.w  D0,-(SP)
                FNC     $02             * putchar
                addq.l  #2,SP

                rts

                .data
msg1:
                dc.b    'ドライブの残りバイト数 : $',0
err_msg:
                dc.b    'ドライブの指定に誤りがあります',13,10
                dc.b    0
crlf_msg:
                dc.b    13,10,0

                .bss
buffer:
                dc.w    4

                .end

*   指定したドライブ（A,B,..）の残り容量を１６進数で表示します。
*
* usage:        sample22 <drive>
```

SMPL 23

```
*
*       Function call sample program #23
*               $FF39 mkdir
*               $FF3B chdir
*

                include macro.h

                .text
start:
                tst.b   (A2)+           * コマンドラインの文字数チェック
                beq     noname          *  0ならばnonameへ

                move.l  A2,cmdline

                move.l  A2,-(SP)        * コマンドライン=ディレクトリ名
                FNC     $39             * mkdir
                addq.l  #4,SP
                tst.l   D0              * エラー?
                bmi     mkdir_err       * [illegible]ならばmkdir_errへ

                move.l  cmdline,-(SP)
                FNC     $3B             * chdir
                addq.l  #4,SP
                tst.l   D0              * エラー?
                bmi     chdir_err

                FNC     $00             * exit
noname:
                pea     noname_msg
                bra     disp_abort
mkdir_err:
                pea     mkdir_err_msg
                bra     disp_abort
chdir_err:
                pea     chdir_err_msg
disp_abort:
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

                .data
noname_msg:
                dc.b    'ディレクトリ名を指定してください',13,10
                dc.b    0
mkdir_err_msg:
                dc.b    'ディレクトリを作成できません',13,10
                dc.b    0
chdir_err_msg:
                dc.b    'ディレクトリを移動できません',13,10
                dc.b    0

                .bss
cmdline:
                ds.l    1

                .end

*       コマンドラインで指定したディレクトリを作成し、その中へ移動します。
*
* usage :       sample23 <pathname>
```

SMPL 24

```
*
*       Function call sample program #24
*               $FF3A rmdir
*               $FF47 curdir
*

                include macro.h

                .text
start:
                pea     buffer
                move.w  #0,-(SP)        * カレントドライブ
                FNC     $47             * curdir
                addq.l  #6,SP

                tst.b   buffer          * ルートディレクトリ?
                beq     root            *   ならばrootへ

                pea     to_mother
                FNC     $3B             * chdir
                addq.l  #4,SP
                tst.l   D0              * エラー?
                bmi     chdir_err       *   ならばchdir_errへ

                pea     path            * 前のカレントディレクトリを削除
                FNC     $3A             * rmdir
                addq.l  #4,SP
                tst.l   D0              * エラー?
                bmi     rmdir_err       *   ならばrmdirへ

                FNC     $00             * exit

root:
                pea     root_msg
                bra     disp_abort
chdir_err:
                pea     chdir_err_msg
                bra     disp_abort
rmdir_err:
                pea     rmdir_err_msg
disp_abort:
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

                .data
to_mother:
                dc.b    '..',0
root_msg:
                dc.b    'ルートディレクトリは削除できません',13,10
                dc.b    0
chdir_err_msg:
                dc.b    'ディレクトリを移動できません',13,10
                dc.b    0
rmdir_err_msg:
                dc.b    'ディレクトリを削除できません',13,10
                dc.b    '(内容が空でないか、発見できない)',13,10
                dc.b    0

path:
                dc.b    '¥'
buffer:
                ds.b    66

                .end
```

```
*	カレントディレクトリから親ディレクトリへ戻り、前のカレントディレク
*	トリを削除します。その際、削除するディレクトリはrmdirコマンドの場合と
*	同様な条件が必要です。
```

SMPL 25

```
*
*	Function call sample program #25
*		$FF3E close
*		$FF3F read
*		$FF40 write
*		$FF42 seek
*

		include macro.h

		.text
start:
		addq.l	#1,A2

		lea	arg1,A1		* argv[1]
		bsr	getarg		* arg取得サブルーチンコール
		tst.l	D0		* argv[1]文字数
		beq	illegal_cmd	*   0ならillegal_cmdへ

		lea	arg2,A1		* argv[2]
		bsr	getarg		* arg取得サブルーチンコール
		tst.l	D0		* argv[2]文字数
		beq	illegal_cmd	*   0ならillegal_cmdへ

		move.w	#2,-(SP)	* READ/WRITE
		pea	arg1
		FNC	$3D		* open
		addq.l	#6,SP
		tst.l	D0		* エラー?
		bmi	open1_err	*   ならばopen1_errへ

		move.w	D0,handl1	* ファイルハンドルをhandl1に保存

		move.w	#0,-(SP)	* READ
		pea	arg2
		FNC	$3D		* open
		addq.l	#6,SP
		tst.l	D0		* エラー?
		bmi	open2_err	*   ならばopen2_errへ

		move.w	D0,handl2	* ファイルハンドルをhandl2に保存

		move.w	#2,-(SP)	* ファイル終端からのオフセット
		move.l	#0,-(SP)	* オフセット0
		move.w	handl1,-(SP)	* ファイル1
		FNC	$42		* seek
		addq.l	#8,SP
		tst.l	D0		* エラー?
		bmi	seek_err	*   ならばseek_errへ
loop:
		move.l	#$8000,-(SP)	* 読みだしサイズ $8000バイト
		pea	buffer		* 読みだしたデータが入るバッファ
		move.w	handl2,-(SP)	* ファイル2
		FNC	$3F		* read
		lea	10(SP),SP
		move.l	D0,D1		* 読みだしバイト数をD1に保存

		move.l	D0,-(SP)	* 書き込みサイズ
		pea	buffer		* 書き込むデータが入っているバッファ
```

```
                move.w  handl1,-(SP)        * ファイル1
                FNC     $40                 * write
                lea     10(SP),SP

                cmp.l   D0,D1               * 読みこんだバイト数と書きこんだバイト数を比較
                bne     write_err           *   等しくなければwrite_errへ
                cmp.l   #$8000,D1           * 読みだしたバイト数をチェック
                beq     loop                *   ファイル終端でなければloopへ

                move.w  handl1,-(SP)        * ファイル1をクローズ
                FNC     $3E                 * close
                addq.l  #4,SP

                move.w  handl2,-(SP)        * ファイル2をクローズ
                FNC     $3E                 * close
                addq.l  #4,SP

                FNC     $00                 * exit

illegal_cmd:
                pea     illegal_cmd_msg
                bra     disp_abort
open1_err:
                pea     open1_err_msg
                bra     disp_abort
open2_err:
                pea     open2_err_msg
                bra     disp_abort
seek_err:
                pea     seek_err_msg
                bra     disp_abort
write_err:
                pea     write_err_msg
disp_abort:
                FNC     $09                 * print
                addq.l  #4,SP

                FNC     $00                 * exit

getarg:                                     * arg取得サブルーチン(簡易型)
                moveq   #0,D0               * 文字数リセット
                moveq   #0,D7               * arg開始フラグリセット
getarg_loop:
                move.b  (A2)+,D1            * 1文字get
                beq     eol                 *   $00ならばeolへブランチ
                cmp.b   #' ',D1             * スペース?
                beq     delim               *   ならばdelimへ
                cmp.b   #9,D1               * タブ?
                beq     delim               *   ならばdelimへ
                                            * それ以外ならば..
                moveq   #-1,D7              *   arg開始フラグをセット
                move.b  D1,(A1)+            * argバッファにコピー & ポインタ+1
                addq.l  #1,D0               * 文字数+1
                bra     getarg_loop
delim:
                tst.l   D7                  * arg開始済み?
                beq     getarg_loop         *   まだならばgetarg_loopへ
eol:
                subq.l  #1,A2               * コマンドラインポインタ補正
                clr.b   (A1)+               * argにエンドマーク

                rts

                .data
illegal_cmd_msg:
                dc.b    'パラメータの指定が間違っています',13,10
                dc.b    0
open1_err_msg:
```

```
		dc.b	'ファイル1をオープンできません',13,10
		dc.b	0
open2_err_msg:
		dc.b	'ファイル2をオープンできません',13,10
		dc.b	0
seek_err_msg:
		dc.b	'ファイル1のseek時にエラーが発生しました',13,10
		dc.b	0
write_err_msg:
		dc.b	'ファイル1のwrite時にエラーが発生しました',13,10
		dc.b	0

		.bss
arg1:
		ds.b	40
arg2:
		ds.b	40
handl1:
		ds.w	1
handl2:
		ds.w	1

buffer:
		ds.b	$8000

		.end

*	コマンドラインで指定した2つのファイル、ファイル1とファイル2につ
*	いて、ファイル1の後ろにファイル2の内容を追加します。テキストファイ
*	ルの場合、ファイル1の終端にEOFコード($1A)が入っているときには、結合
*	後TYPEしてみてもファイル1の内容しか表示されませんので、DUMPで結合を
*	確認してみてください。
*
*	usage:		sample25 <file1> <file2>
```

SMPL 26

```
*
*	Function call sample program #26
*		$FF37 nameck
*		$FF41 delete
*		$FF4E files
*		$FF4F nfiles
*

		include	macro.h

		.text
start:
		addq.l	#1,A2

		lea	arg1,A1		* argv[1] : ファイルマスク
		bsr	getarg		* arg取得サブルーチンコール
		tst.l	D0		* argv[1]文字数
		beq	illegal_cmd	*   0ならillegal_cmdへ

		pea	namebuf
		pea	arg1
		FNC	$37		* nameck
		addq.l	#8,SP

		lea	namebuf,A0
nulsrc:
		tst.b	(A0)+
		bne	nulsrc
```

```
        subq.l  #1,A0
        move.l  A0,nameptr

        lea     arg2,A1         * argv[2] : 日付
        bsr     getarg          * arg取得サブルーチンコール
        tst.l   D0              * argv[2]文字数
        beq     illegal_cmd     *   0ならillegal_cmdへ

        lea     arg2,A1
        bsr     ascdatebin2     * 日付(ASCII)→バイナリルーチンコール
        tst.l   D0              * エラー?
        bmi     date_err        *   ならばdate_errへ
        move.w  D0,date         * 日付の内容をdateに保存

        move.w  #0,time         * 時刻00:00:00
        lea     arg3,A1         * argv[3] : 時刻
        bsr     getarg          * arg取得サブルーチンコール
        tst.l   D0              * argv[3]文字数
        beq     lbl1            *   0ならlbl1へ

        lea     arg3,A1
        bsr     asctimebin2     * 時刻(ASCII)→バイナリルーチンコール
        tst.l   D0              * エラー?
        bmi     time_err        *   ならばtime_errへ
        move.w  D0,time         * 時刻の内容をtimeへ
lbl1:
        move.w  #0,filenum      * マッチしたファイル数をリセット
        move.w  #$20,-(SP)      * 通常ファイル
        pea     arg1            * arg[1]
        pea     filbuf          * 結果が返るバッファ
        FNC     $4E             * files
        lea     10(SP),SP
        tst.l   D0              * 該当するファイルはあるか?
        bmi     nomatch         *   ないならばnomatchへ

        bra     lbl2

loop:
        pea     filbuf          * 結果が返るバッファ
        FNC     $4F             * nfiles
        addq.l  #4,SP
        tst.l   D0              * これ以上該当するファイルはあるか?
        bmi     nomatch         *   ないならばnomatchへ
lbl2:
        lea     filbuf,A1
        move.w  22(A1),D0       * ファイルの時刻
        move.w  24(A1),D1       * ファイルの日付
        cmp.w   date,D1         * 指定された日付とチェック
        bcs     loop            *   dateより前ならloopに戻る
        cmp.w   time,D0         * 指定された時刻とチェック
        bcs     loop            *   timeより前ならloopに戻る
lbl3:
        add.w   #1,filenum      * マッチしたファイル数+1

        pea     30(A1)          * PACKEDNAME
        FNC     $09             * print
        addq.l  #4,SP
        pea     delyn_msg
        FNC     $09             * print
        addq.l  #4,SP

        FNC     $01             * getchar
        move.l  D0,D1

        pea     crlf_msg
        FNC     $09             * print
        addq.l  #4,SP

        or.b    #$20,D1         * 大文字に対応する処理
```

```
        cmp.b   #'y',D1         * "y"か？
        beq     delete          * 〃ならばdeleteへ
        cmp.b   #'n',D1         * "n"か？
        beq     loop            * 〃ならばloopへ

        bra     lbl3            * それ以外なら再び尋ねる
delete:
        move.l  nameptr,A0
        lea     30(A1),A1
        move.w  #22,D0
namecopy:
                                * パス名とファイル名を結合
        move.b  (A1)+,(A0)+
        dbra    D0,namecopy

        pea     namebuf
        FNC     $41             * delete
        addq.l  #4,SP
        tst.l   D0
        bmi     delete_err

        bra     loop
nomatch:
        tst.w   filenum
        beq     nofile
        pea     nomore_msg
        bra     lbl4
nofile:
        pea     nofile_msg
lbl4:
        FNC     $09             * print
        addq.l  #4,SP

        FNC     $00             * exit

illegal_cmd:
        pea     illegal_cmd_msg
        bra     disp_abort
date_err:
        pea     date_err_msg
        bra     disp_abort
time_err:
        pea     time_err_msg
        bra     disp_abort
delete_err:
        pea     delete_err_msg
disp_abort:
        FNC     $09             * print
        addq.l  #4,SP

        FNC     $00             * exit
getarg:                         * arg取得サブルーチン(簡易型)
        moveq   #0,D0           * 文字数リセット
        moveq   #0,D7           * arg開始フラグリセット
getarg_loop:
        move.b  (A2)+,D1        * 1文字get
        beq     eol             * 〃 $00ならばeolへブランチ
        cmp.b   #' ',D1         * スペース？
        beq     delim           * 〃ならばdelimへ
        cmp.b   #9,D1           * タブ？
        beq     delim           * 〃ならばdelimへ
                                * それ以外ならば．．
        moveq   #-1,D7          *   arg開始フラグをセット
        move.b  D1,(A1)+        * argバッファにコピー & ポインタ＋1
        addq.l  #1,D0           * 文字数＋1
        bra     getarg_loop
delim:
        tst.l   D7              * arg開始済み？
        beq     getarg_loop     *   まだならばgetarg_loopへ
```

```
eol:
                subq.l  #1,A2           * コマンドラインポインタ補正
                clr.b   (A1)+           * argにエンドマーク

                rts

ascdatebin2:                            * 日付(ASCII)→バイナリルーチンコール
                IOCS    $58             * DATECNV (IOCS)
                tst.l   D0
                bmi     adb2_exit
                move.l  D0,D1
                move.l  D0,D2
                and.l   #%000000000000_00000000_00011111,D0     * day
                and.l   #%000000000000_00001111_00000000,D1     * month
                lsr.l   #3,D1
                and.l   #%111111111111_00000000_00000000,D2     * year
                sub.l   #$7BC_00_00,D2
                lsr.l   #7,D2
                or.l    D1,D0
                or.l    D2,D0
adb2_exit:
                rts

asctimebin2:                            * 時刻(ASCII)→バイナリルーチンコール
                IOCS    $59             * TIMECNV (IOCS)
                tst.l   D0
                bmi     atb2_exit
                move.l  D0,D1
                move.l  D0,D2
                and.l   #%00000000_00000000_00111110,D0 * sec
                lsr.l   #1,D0
                and.l   #%00000000_00111111_00000000,D1 * min
                lsr.l   #3,D1
                and.l   #%00011111_00000000_00000000,D2 * hour
                lsr.l   #5,D2
                or.l    D1,D0
                or.l    D2,D0
atb2_exit:
                rts

                .data
illegal_cmd_msg:
                dc.b    'パラメータの指定が間違っています',13,10
                dc.b    0
date_err_msg:
                dc.b    '日付の指定が間違っています',13,10
                dc.b    0
time_err_msg:
                dc.b    '時刻の指定が間違っています',13,10
                dc.b    0
delete_err_msg:
                dc.b    'デリートできませんでした',13,10
                dc.b    0
delyn_msg:
                dc.b    ' DELETE (Y/N) ? ',0
nomore_msg:
                dc.b    '以上です',13,10
                dc.b    0
nofile_msg:
                dc.b    '該当するファイルはありません',13,10
                dc.b    0
crlf_msg:
                dc.b    13,10,0

                .bss
namebuf:
                ds.b    92
filbuf:
                ds.b    53
```

```
arg1:
		ds.b	40
arg2:
		ds.b	40
arg3:
		ds.b	40
		.even
time:
		ds.w	1
date:
		ds.w	1
filenum:
		ds.w	1
nameptr:
		ds.l	1

		.end

*   コマンドラインでファイルマスク、日付、時刻を指定し、ファイルマスク
* に一致するファイル群の中の、指定した日付／時刻以降のものを消去します。
* 消去する前には確認を求めてきます。時刻は省略可能です。
*
* usage:	sample26 <mask> <date> [<time>]
```

SMPL 27

```
*
*	Function call sample program #27
*		$FF43 chmod
*

		include macro.h

		.text
start:
		tst.b	(A2)+		* コマンドラインの文字数をチェック
		beq	noname		*   0ならnonameへ

		move.w	#-1,-(SP)	* 現在のモードを得る
		move.l	A2,-(SP)
		FNC	$43		* chmod
		addq.l	#6,SP
		tst.l	D0		* エラー？
		bmi	nofile		*   ならばnofileへ

		eor.w	#$0001,D0	* Read/Only属性を反転

		move.w	D0,-(SP)
		move.l	A2,-(SP)
		FNC	$43		* chmod
		add.l	#6,SP

		FNC	$00		* exit

noname:
		pea	noname_msg
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00		* exit

nofile:
		pea	nofile_msg
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00		* exit
```

```
		.data
noname_msg:
		dc.b	'ファイルを指定してください',13,10
		dc.b	0
nofile_msg:
		dc.b	'ファイルがみつかりません',13,10
		dc.b	0

		.end

*	コマンドラインで指定したファイルのRead/Only属性を切り替えます。
*
* usage:		sample27 <filename>
```

SMPL 28

```
*
*	Function call sample program #28
*		$FF44 ioctrl
*

		include macro.h

		.text
start:
		move.w	#$02,-(SP)		* READ/WRITE
		pea	pcmname			* PCMデバイスをオープン
		FNC	$3D			* open
		addq.l	#6,SP

		move.l	D0,D1			* ハンドルをD2に保存

		move.l	#1,-(SP)	* PCMﾃﾞﾊﾞｲｽのこのioctrlでは意味を持たない
		pea	buffer			* 値の返るバッファを指すポインタ
		move.w	D1,-(SP)		* ハンドル
		move.w	#2,-(SP)		* デバイスドライバからの直接入力
		FNC	$44			* ioctrl
		lea	12(SP),SP		* 現在のPCMﾃﾞﾊﾞｲｽの実行状況が返る

		move.w	D1,-(SP)
		FNC	$3E			* close
		addq.l	#2,SP

		moveq	#0,D0
		move.b	buffer,D0
		beq	nothing
		cmp.b	#$20,D0
		bcs	lbl1
		sub.b	#$0C,D0
lbl1:
		cmp.b	#$10,D0
		bcs	lbl2
		sub.b	#$0C,D0
lbl2:
		subq	#2,D0
		lsl.w	#1,D0
		lea	msg_table,A0
		move.l	(A0,D0.w),-(SP)
		FNC	$09			* print
		addq.l	#4,SP

		FNC	$00			* exit

nothing:
		pea	nothing_msg
		FNC	$09			* print
```

```
		addq.l	#4,SP

		FNC	$00			* exit

		.data
pcmname:
		dc.b	'PCM',0
nothing_msg:
		dc.b	'何も実行していません。',13,10,0
		.even
msg_table:
		dc.l	msg02,msg04,msg12,msg14,msg22,msg24
msg02:
		dc.b	'出力中：IOCSコール$60を実行中。',13,10,0
msg04:
		dc.b	'入力中：IOCSコール$61を実行中。',13,10,0
msg12:
		dc.b	'出力中：IOCSコール$62を実行中。',13,10,0
msg14:
		dc.b	'入力中：IOCSコール$63を実行中。',13,10,0
msg22:
		dc.b	'出力中：IOCSコール$64を実行中。',13,10,0
msg24:
		dc.b	'入力中：IOCSコール$65を実行中。',13,10,0

		.bss
buffer:
		ds.b	1

		.end

*	IOCTRL可能なデバイス"PCM"を利用して、現在のPCMの実行状況を表示しま
* す。
```

SMPL 29

```
*
*	Function call sample program #29
*		$FF2F dup0
*

		include	macro.h

		.text
start:
		tst.b	(A2)+			* コマンドラインの文字数チェック
		beq	noname			* 0ならばnonameへ

		move.w	#$20,-(SP)		* 通常ファイル
		move.l	A2,-(SP)		* コマンドライン＝ファイル名
		FNC	$3C			* create
		addq.l	#6,SP
		tst.l	D0			* エラー？
		bmi	create_err		* ならばcreate_errへ

		move.w	D0,handl		* ファイルハンドルをhandlへ

		move.w	#2,-(SP)		* 標準エラー出力
		move.w	D0,-(SP)
		FNC	$2F			* dup0
		addq.l	#4,SP
		tst.l	D0			* エラー？
		bmi	dup0_err		* ならばdup0_errへ

		move.l	#BOTTOM-start+$F0,-(SP) * 自分に必要最小限のメモリ
		pea	16(A0)			* 現在のPSP
```

```
		FNC	$4A		* setblock
		addq.l	#8,SP

		clr.l	-(SP)		* 現在の環境
		pea	P1		* コマンドライン: (null)
		pea	FILE		* ファイル名	: a:¥command.x
		move.w	#0,-(SP)	* load & exec
		FNC	$4B		* exec
		lea	14(SP),SP
		tst.l	D0		* エラー?
		bmi	exec_err	*  〃ならばexec_errへ

		move.w	#2,-(SP)	* 標準エラー出力
		move.w	#2,-(SP)	* もとに戻す
		FNC	$2F		* dup0
		addq.l	#4,SP

		move.w	handl,-(SP)
				* もとのﾌｧｲﾙをｸﾛｰｽﾞ(dup0の返り値を渡してもよい)
		FNC	$3E		* close
		addq.l	#2,SP

		FNC	$00		* exit

noname:
		pea	noname_msg
		bra	disp_abort
create_err:
		pea	create_err_msg
		bra	disp_abort
dup0_err:
		pea	dup0_err_msg
		bra	disp_abort
exec_err:
		pea	exec_err_msg
disp_abort:
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00		* exit

		.data
noname_msg:
		dc.b	'ファイル名を指定してください',13,10
		dc.b	0
create_err_msg:
		dc.b	'ファイルを作成できません',13,10
		dc.b	0
dup0_err_msg:
		dc.b	'ファイルハンドルを強制複写できません',13,10
		dc.b	0
exec_err_msg:
		dc.b	'command.xを起動できません',13,10
		dc.b	0

P1:
		dc.b	0
FILE:
		dc.b	'a:¥command.x',0

		.bss
handl:
		ds.w	1

BOTTOM:		equ	*

		.end
```

```
*  標準エラー出力を、コマンドラインで指定したファイルへリダイレクトし
* た状態でcommand.xを起動します。その結果、このcommand.xから立ち上げた
* プログラムが標準エラー出力に出力した文字列は、指定したファイルの中に
* 入ることになります。
*
* usage:          sample29 <filename>
```

SMPL 30

```
*
*       Function call sample program #30
*               $FF45 dup
*               $FF46 dup2
*

                include macro.h

                .text
start:
                move.w  #1,-(SP)                * 標準出力
                FNC     $45                     * dup
                addq.l  #2,SP
                tst.l   D0                      * エラー?
                bmi     dup_err                 *   ならばdup_errへ

                move.w  D0,handl1               * 複写先のハンドルを保存

                addq.w  #1,D0                   * 次のハンドル
                move.w  D0,handl2               * 複写先のハンドルを保存
                move.w  D0,-(SP)                * 次のハンドルに複写
                move.w  #1,-(SP)                * 標準出力
                FNC     $46                     * dup2
                addq.l  #4,SP
                tst.l   D0
                bmi     dup2_err

                move.w  handl1,D1
                move.w  D1,-(SP)                * 複写ハンドル1に出力
                pea     msg1
                FNC     $1E                     * fputs
                addq.l  #6,SP
                move.l  D1,D0
                bsr     hex2                    * 16進2桁表示サブルーチン
                move.w  D1,-(SP)                * 複写ハンドル1に出力
                pea     msg2
                FNC     $1E                     * fputs
                addq.l  #6,SP

                move.w  handl2,D1
                move.w  D1,-(SP)                * 複写ハンドル2に出力
                pea     msg1
                FNC     $1E                     * fputs
                addq.l  #4,SP
                move.l  D1,D0
                bsr     hex2                    * 16進2桁表示サブルーチン
                move.w  D1,-(SP)                * 複写ハンドル2に出力
                pea     msg2
                FNC     $1E                     * fputs
                addq.l  #6,SP
                move.w  handl1,-(SP)
                FNC     $3E                     * close
                addq.l  #4,SP

                move.w  handl2,-(SP)
                FNC     $3E                     * close
                addq.l  #4,SP
```

```
		FNC	$00		* exit

dup_err:
		pea	dup_err_msg
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00		* exit
dup2_err:
		pea	dup2_err_msg
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00		* exit

					* 16進2桁表示サブルーチン
hex2:
		move.l	D0,-(SP)	* (出力ハンドル指定)
		lsr.w	#4,D0
		bsr	hex1
		move.l	(SP)+,D0
hex1:
		and.w	#$0F,D0
		add.w	#'0',D0
		cmp.w	#$3A,D0
		bcs	hex0
		add.w	#7,D0
hex0:
		move.w	D1,-(SP)
		move.w	D0,-(SP)
		FNC	$1D		* fputc
		addq.l	#4,SP

		rts

		.data
msg1:
		dc.b	'ファイルハンドル ',0
msg2:
		dc.b	' に出力しています',13,10
		dc.b	0
dup_err_msg:
		dc.b	'ファイルを複写(dup)できません',13,10
		dc.b	0
dup2_err_msg:
		dc.b	'ファイルを強制複写(dup2)できません',13,10
		dc.b	0

		.bss
handl1:
		ds.w	1
handl2:
		ds.w	1

		.end
```

* 標準出力を複写／強制複写し、それぞれの複写先のファイルハンドルを使っ
* て、標準出力に設定されていたデバイスに文字列を出力してみせます。

SMPL 31

a) "SAMPLE31.S"

```
*
*	Function call sample program #31 part1
*		$FF4B exec
*		$FF4D wait
*
```

```
		include	macro.h

		.text
start:
		move.l	#BOTTOM-start+$F0,-(SP) * 自分に必要最小限のメモリ
		pea	16(A0)		* 現在のPSP
		FNC	$4A		* setblock
		addq.l	#8,SP

		clr.l	-(SP)		* 環境は現在のものを引き継ぐ
		pea	P1		* コマンドラインのポインタ
		pea	FILE		* ファイルネーム
		move.w	#0,-(SP)	* モード0 : LOAD & EXEC
		FNC	$4B		* exec
		lea	14(SP),SP
		tst.l	D0		* エラー？
		bmi	cannot_exec	*  ならばcannot_execへ

		FNC	$4D		* wait

		lsl.w	#2,D0		* 終了コード＊4
		lea	msgtabl,A0
		move.l	(A0,D0.w),-(SP)
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00		* exit

cannot_exec:
		pea	cannot_exec_msg
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00		* exit

P1:
		dc.b	0
FILE:
		dc.b	's31_2.x',0
		.even
msgtabl:
		dc.l	msg1,msg2,msg3
msg1:
		dc.b	'パワースイッチにより起動しました',13,10
		dc.b	0
msg2:
		dc.b	'外部スイッチにより起動しました',13,10
		dc.b	0
msg3:
		dc.b	'タイマにより起動しました',13,10
		dc.b	0

cannot_exec_msg:
		dc.b	's31_2.xを実行できませんでした',13,10
		dc.b	0

BOTTOM:		equ	*

		.end
```

b) "S31_2.S"

```
*
*	Function call sample program #31 part2
*		$FF4C exit2
*

		include	macro.h
```

```
		.text
start:
		IOCS	$8E		* BOOTINF (IOCS)

		lsr.l	#8,D0
		lsr.l	#8,D0
		lsr.l	#8,D0		* 起動情報を得る

		move.w	D0,-(SP)
		FNC	$4C		* exit2

		.end

*  このサンプルは２つのプログラムを１組として使用してください。
* 【part1について】
*   ＄ＦＦ４Ｂexecでpart2をロード＆実行しています。終了コードはexecコー
* ルの返り値としてD0に返ってきますが、ここではあえて＄ＦＦ４Ｄwaitを使
* 用して再度終了コードを得ています。
* 【part2について】
*   ＩＯＣＳを利用して得られた起動情報を終了コードとして返しています。
*
*  part1とpart2は同じディレクトリにある必要があります。
```

SMPL 32

```
*
*	Function call sample program #32
*		$FF52 setenv
*		$FF53 getenv
*

		include macro.h

start:
		tst.b	(A2)+		* コマンドラインの文字数チェック
		beq	noname		*   0ならばnonameへ
		move.l	A2,cmdptr

		pea	getbuf
		clr.l	-(SP)
		pea	getname		* 環境変数'path'
		FNC	$53		* getenv
		lea	12(SP),SP
		tst.l	D0		* エラー？
		bmi	get_err		*   ならばget_errへ

		lea	getbuf,A1	* バッファの先頭からサーチ開始
		move.l	A1,begin_ptr
		move.l	cmdptr,A2	* コマンドラインの内容をサーチ
		clr.b	skipping	* スキップ中フラグクリア
		clr.b	eol_flag	* 行末フラグクリア
search:
		move.b	(A1)+,D0	* バッファから１文字
		beq	eol		*   $00なら行末。eolへ
		cmp.b	#';',D0		* デリミタ';'か？
		beq	delim		*   ならばdelimへ
		tst.b	skipping	* スキップ中？
		bne	search		*   ならばsearchへ
		cmp.b	(A2)+,D0	* コマンドラインと比較
		beq	search		*   一致するならマッチングを続ける

		move.b	#-1,skipping	* スキップ開始
		bra	search

eol:
		move.b	#-1,eol_flag	* 行末フラグを立てる
		sub.l	#1,begin_ptr	* 最後の';'を消すため
```

```
delim:
                tst.b   skipping        * スキップ中だった？
                bne     delim2          *   ならばdelim2へ
                tst.b   (A2)            * 比較文字列も行末？
                beq     match           *   ならば完全一致でmatchへ
delim2:
                tst.b   eol_flag        * 行末だった？
                bne     not_found       *   ならばnot_foundへ
                move.l  A1,begin_ptr    * ポインタ更新
                move.l  cmdptr,A2       * 再びコマンドライン先頭からマッチング
                clr.b   skipping        * スキップ中フラグはクリア
                bra     search
match:
                move.w  #255,D0         * 最高でも２５５文字まで
                move.l  begin_ptr,A2
match_loop:                             * 削除開始
                move.b  (A1)+,(A2)+
                dbeq    D0,match_loop

                pea     getbuf          * 処理後の変数の内容
                clr.l   -(SP)
                pea     getname         * 環境変数'path'
                FNC     $52             * setenv
                addq.l  #8,SP
                tst.l   D0              * エラー？
                bmi     set_err         *   ならばset_errへ

                FNC     $00             * exit

not_found:
                pea     not_found_msg
                bra     disp_abort
noname:
                pea     noname_msg
                bra     disp_abort
get_err:
                pea     get_err_msg
                bra     disp_abort
set_err:
                pea     set_err_msg
disp_abort:
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

                .data
not_found_msg:
                dc.b    '指定されたパスは登録されていません',13,10
                dc.b    0
noname_msg:
                dc.b    '削除するパス名を指定してください',13,10
                dc.b    0
get_err_msg:
                dc.b    '環境変数',$27,'path',$27,'がありません',13,10
                dc.b    0
set_err_msg:
                dc.b    '環境変数',$27,'path',$27,'を設定できません',13,10
                dc.b    0

getname:
                dc.b    'path',0

                .bss
cmdptr:
                ds.l    1
begin_ptr:
                ds.l    1
```

```
getbuf:
                ds.b    256
skipping:
                ds.b    1

                .end

*   環境変数'path'に登録されているパスの中から、コマンドラインで指定し
*   たパスを削除します。
*
*   usage:      sample32 <path>
```

SMPL 33

```
*
*       Function call sample program #33
*               $FF2E verify
*               $FF54 verifyg
*

                include macro.h

                .text
start:
                FNC     $54             * verifyg
                tst.l   D0              * ベリファイしている?
                bne     turn_off        *   しているならばturn_offへ
turn_on:
                move.w  #1,D0           * VERIFY = ON
                lea     verify_on_msg,A1
                bra     set_verify
turn_off:
                move.w  #0,D0           * VERIFY = OFF
                lea     verify_off_msg,A1
set_verify:
                move.w  D0,-(SP)
                FNC     $2E             * verify
                addq.l  #2,SP

                move.l  A1,-(SP)
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

                .data
verify_on_msg:
                dc.b    'VERIFY を <on> にしました',13,10
                dc.b    0
verify_off_msg:
                dc.b    'VERIFY を <off> にしました',13,10
                dc.b    0

                .end

*   VERIFYの<on>、<off>を切り替えます。
```

SMPL 34

```
*
*       Function call sample program #34
*               $FF56 rename
*

                include macro.h

                .text
start:
                tst.b   (A2)+           * コマンドラインの文字数チェック
                beq     noname          *   0ならばnonameへ

                lea     arg1,A1         * argv[1]
                bsr     getarg          * arg取得サブルーチンコール
                tst.l   D0              * 文字数は0？
                beq     arg_err         *   ならばarg_errへ

                lea     arg2,A1         * argv[2]
                bsr     getarg          * arg取得サブルーチンコール
                tst.l   D0              * 文字数は0？
                beq     arg_err         *   ならばarg_errへ

                pea     arg2            * 変更後の名前
                pea     arg1            * 変更前の名前
                FNC     $56             * rename
                addq.l  #8,SP
                tst.l   D0              * エラー？
                bmi     ren_err

                FNC     $00             * exit

ren_err:
                pea     ren_err_msg
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

noname:
                pea     noname_msg
                bra     disp_abort
arg_err:
                pea     arg_err_msg
disp_abort:
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit
getarg:                                 * arg取得サブルーチン(簡易型)
                moveq   #0,D0           * 文字数リセット
                moveq   #0,D7           * arg開始フラグリセット
getarg_loop:
                move.b  (A2)+,D1        * 1文字get
                beq     eol             *   $00ならばeolへブランチ
                cmp.b   #' ',D1         * スペース？
                beq     delim           *   ならばdelimへ
                cmp.b   #9,D1           * タブ？
                beq     delim           *   ならばdelimへ
                                        * それ以外ならば..
                moveq   #-1,D7          *   arg開始フラグをセット
                move.b  D1,(A1)+        * argバッファにコピー & ポインタ+1
                addq.l  #1,D0           * 文字数+1
                bra     getarg_loop
delim:
                tst.l   D7              * arg開始済み？
                beq     getarg_loop     *   まだならばgetarg_loopへ
```

```
eol:
                subq.l  #1,A2           * コマンドラインポインタ補正
                clr.b   (A1)+           * argにエンドマーク

                rts

                .data
noname_msg:
                dc.b    '旧、新各ファイル名を指定してください',13,10
                dc.b    0
arg_err_msg:
                dc.b    'パラメータが異常です',13,10
                dc.b    0
ren_err_msg:
                dc.b    'リネームに失敗しました',13,10
                dc.b    0

                .bss
arg1:
                ds.b    64
arg2:
                ds.b    64

                .end.

*   コマンドラインで指定した旧、新各ファイル名に従ってリネームを行いま
* す。COMMAND.XのRENコマンドとは、新旧で異なるディレクトリを指定した場
* 合、ファイルの移動も行う点、そしてディレクトリ名のリネームも行える点
* です。COMMAND.Xでは、あえてファンクションコール$FF56 renameの機能
* を制限して利用しています。
*
* usage:        sample34 <old_name> <new_name>
```

SMPL 35

```
*
*       Function call sample program #35
*               $FF5A creattmp
*

                include macro.h

                .text
start:
                move.w  #3-1,D7         * 3個ファイルを作る
loop:
                move.w  #$20,-(SP)      * 通常ファイル
                pea     name
                FNC     $5A             * creattmp
                addq.l  #6,SP
                tst.l   D0
                beq     open_err

                move.w  D0,-(SP)
                FNC     $3E             * close
                addq.l  #2,SP

                dbra    D7,loop

                FNC     $00             * exit

open_err:
                pea     open_err_msg
                FNC     $09             * print
                addq.l  #4,SP
```

```
                FNC     $00             * exit


                .data
name:
                dc.b    'temp????.$$$',0
open_err_msg:
                dc.b    'これ以上テンポラリファイルをオープンできません',13,10
                dc.b    0

                .end

*   'tmp????.$$$'というマスクで表現されるテンポラリファイルを3個クリエ
*   イトします。CONFIG.SYSで"SHARE="を指定している場合、第1パラ
*   メータは3以上にしてください。
*   複数回続けて実行してみると、動作がよく理解できると思います。
```

SMPL 36

```
*
*       Function call sample program #36
*               $FF5B create2
*

                include macro.h

                .text
start:
                tst.b   (A2)+           * コマンドラインの文字数チェック
                beq     nofile          *   0ならばnofileへ

                move.w  #$20,-(SP)      * 通常ファイル
                move.l  A2,-(SP)        * コマンドライン=ファイル名
                FNC     $5B             * cteate2
*               FNC     $3C             * create
                addq.l  #6,SP
                tst.l   D0              * エラー?
                bmi     error           *   ならばerrorへ

                move.w  D0,-(SP)
                FNC     $3E             * close
                addq.l  #2,SP

                pea     create_msg
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

error:
                cmp.l   #-$80,D0        * 既に同名のファイルが存在?
                bne     open_err

                pea     exist_msg
                bra     disp_abort
open_err:
                pea     open_err_msg
                bra     disp_abort
nofile:
                pea     nofile_msg
disp_abort:
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit
```

```
		.data
nofile_msg:
		dc.b	'ファイル名を指定してください',13,10
		dc.b	0
create_msg:
		dc.b	'ファイルを作成しました',13,10
		dc.b	0
exist_msg:
		dc.b	'ファイルは既に存在します',13,10
		dc.b	0
open_err_msg:
		dc.b	'ファイルを作成できません',13,10
		dc.b	0

		.end

*	コマンドラインで指定したファイル名のファイルを作成します。既に存在
* するファイルと同名のファイルを作成しようとした場合、作成せずに終了し
* ます。＄ＦＦ５Ｂcreate2を使用している部分を＄ＦＦ３Ｃcreateに置き換え
* て違いを確認してください。
*
* usage:	sample36 <filename>
```

SMPL 37

```
*
*	Function call sample program #37
*		$FF5C lock
*

		include macro.h

		.text
start:
		tst.b	(A2)+		* コマンドラインの文字数チェック
		beq	noname		*   0ならばnonameへ

		move.w	#0,-(SP)	* Compatible/READ
		move.l	A2,-(SP)	* コマンドライン＝ファイル名
		FNC	$3D		* open
		addq.l	#6,SP
		tst.l	D0		* エラー?
		bmi	open_err	*   ならばopen_errへ

		move.w	D0,handl	* ファイルハンドルをhandlへ

		move.w	#2,-(SP)	* ファイルの終端から．．
		move.l	#0,-(SP)	* オフセット０の地点へ
		move.w	handl,-(SP)
		FNC	$42		* seek
		addq.l	#8,SP		* D0にはファイルサイズ

		move.l	D0,filesize	* ファイルサイズをfilesizeへ

		move.l	D0,-(SP)	* ファイル全域を．．
		move.l	#0,-(SP)	* ファイル先頭からロックします
		move.w	handl,-(SP)
		move.w	#0,-(SP)	* ロック設定
		FNC	$5C		* lock
		lea	12(SP),SP

		pea	lock_msg
		FNC	$09		* print
		addq.l	#4,SP
```

```
                move.l  #BOTTOM-start+$F0,-(SP) * 自分に必要最小限のメモリ
                pea     16(A0)          * 現在のPSP
                FNC     $4A             * setblock
                addq.l  #8,SP

                clr.l   -(SP)           * 現在の環境
                pea     P1              * コマンドライン: (null)
                pea     FILE            * ファイル名    : a:¥command.x
                move.w  #0,-(SP)        * load & exec
                FNC     $4B             * exec
                lea     14(SP),SP
                move.l  D0,-(SP)        * execの結果をプッシュ

                move.l  filesize,-(SP)  * ファイル全域を..
                move.l  #0,-(SP)        * ファイル先頭からロックします
                move.w  handl,-(SP)
                move.w  #1,-(SP)        * ロック解除
                FNC     $5C             * lock
                lea     12(SP),SP

                move.l  (SP)+,D0        * execの結果をポップ
                tst.l   D0              * エラー?
                bmi     exec_err        *   ならばexec_errへ

                pea     unlock_msg
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

noname:
                pea     noname_msg
                bra     disp_abort
open_err:
                pea     open_err_msg
                bra     disp_abort
exec_err:
                pea     exec_err_msg
disp_abort:
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

                .data
P1:
                dc.b    0
                dc.b    0
FILE:
                dc.b    'a:¥command.x',0

lock_msg:
                dc.b    'ファイルをロックしました',13,10
                dc.b    0
unlock_msg:
                dc.b    'ファイルのロックを解除しました',13,10
                dc.b    0

noname_msg:
                dc.b    'ファイル名を指定してください',13,10
                dc.b    0
open_err_msg:
                dc.b    'ファイルがみつかりません',13,10
                dc.b    0
exec_err_msg:
                dc.b    'command.xがみつかりません',13,10
                dc.b    0
```

```
		.bss
handl:
		ds.w	1
filesize:
		ds.l	1

BOTTOM:		equ	*

		.end

*	コマンドラインで指定したファイル全域をロックした状態でcommand.xを起
*	動します。TYPEコマンドなどを使って、指定したファイルにアクセスできな
*	いことを確認してください。
*	このプログラムを実行するためには、ドライブAのルートディレクトリに
*	command.xが存在することが必要です。
*
*	usage:		sample37 <filename>
```

SMPL 38

```
*
*	Function call sample program #38
*		$FF5F assign
*

		include	macro.h

		.text
start:
		tst.b	(A2)+		* コマンドラインの文字数をチェック
		beq	noname		*   文字数0ならnodrvへ

		pea	buffer
		move.l	A2,-(SP)	* コマンドライン=ドライブネーム
		move.w	#0,-(SP)	* アサイン状況を得る
		FNC	$5F		* assign
		lea	10(SP),SP
		tst.l	D0
		bmi	drverr

		cmp.b	#$40,D0		* アサインインデックスが$40なら
		beq	normal		*   normalへ
		cmp.b	#$50,D0		* アサインインデックスが$50なら
		beq	vertical	*   verticalへ
		cmp.b	#$60,D0		* アサインインデックスが$60なら
		beq	subdir		*   subdirへ

		bra	nodrv

normal:
		pea	normal_msg
		bra	lbl1
vertical:
		pea	vertical_msg
		bra	lbl1
subdir:
		pea	subdir_msg
lbl1:
		FNC	$09		* print
		addq.l	#4,SP

		pea	buffer
		FNC	$09		* print
		addq.l	#4,SP

		pea	desu
```

```
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit


                pea     noname_msg
                bra     disp_abort


                pea     drverr_msg
                bra     disp_abort


                pea     nodrv_msg
_abort:
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

                .data
e_msg:
                dc.b    'ドライブネームを指定してください',13,10
                dc.b    0
_msg:
                dc.b    'ドライブネームが異常です',13,10
                dc.b    0
_msg:
                dc.b    'ドライブが存在しません',13,10
                dc.b    0
al_msg:
                dc.b    'ドライブは通常のドライブで、カレントディレクトリは',0
ical_msg:
                dc.b    'ドライブは仮想ドライブで、このドライブとしてアクセスする'
                dc.b    'サブディレクトリは',0
dir_msg:
                dc.b    'ドライブは他のドライブのサブディレクトリとしてアクセス'
                dc.b    'され、そのディレクトリは',0
desu:
                dc.b    'です。',13,10
                dc.b    0

                .bss
ffer:
                ds.b    65

                .end

*       コマンドラインで指定したドライブのアサイン状況を調べます。
*
* usage:        sample38 <drive>
```

SMPL 39

```
*
*       Function call sample program #39
*               $FF7C getfcb
*

                include macro.h

                .text
start:
                tst.b   (A2)+           * コマンドラインの文字数をチェック
                beq     noname          *   0ならnonameへ

                clr.l   -(SP)           * スーパーバイザモードへ
```

```
        FNC     $20             * super
        addq.l  #4,SP           * 返り値は読み捨てる
                                * (ユーザモードへの復帰を省略したため)
        move.w  #0,-(SP)        * READ
        move.l  A2,-(SP)
        FNC     $3D             * open
        addq.l  #4,SP
        tst.l   D0              * エラー?
        bmi     open_err        * ならばopen_errへ

        move.l  D0,D7           * ファイルハンドルをD7に保存

        move.w  D0,-(SP)
        FNC     $7C             * getfcb
        addq.l  #4,SP

        move.l  D0,A1
        move.w  #$60-1,D1
loop:
        move.b  (A1)+,D0
        bsr     hex2            * 16進2桁表示ルーチンコール
        move.w  #' ',-(SP)
        FNC     $02             * putchar
        addq.l  #2,SP

        dbra    D1,loop

        move.w  D7,-(SP)
        FNC     $3E             * close
        addq.l  #2,SP

        FNC     $00             * exit

noname:
        pea     noname_msg
        FNC     $09             * print
        addq.l  #4,SP
        FNC     $00             * exit

open_err:
        pea     open_err_msg
        FNC     $09             * print
        addq.l  #4,SP

        FNC     $00             * exit

hex2:                           * 16進2桁表示ルーチン
        move.l  D0,-(SP)
        lsr.w   #4,D0
        bsr     hex1
        move.l  (SP)+,D0
hex1:
        and.w   #$0F,D0
        add.w   #'0',D0
        cmp.w   #$3A,D0
        bcs     hex0
        add.w   #7,D0
hex0:
        move.w  D0,-(SP)
        FNC     $02             * putchar
        addq.l  #2,SP

        rts


        .data
noname_msg:
        dc.b    'ファイル名を指定してください',13,10
```

```
	dc.b	0
_err_msg:
	dc.b	'ファイルがみつかりません',13,10
	dc.b	0

	.end
```

コマンドラインで指定したファイルをオープンし、そのＦＣＢの内容をダンプします。

```
usage:		sample39 <filename>
```

SMPL 40

```
*
*	Function call sample program #40
*		$FF7D malloc2
*		$FF7E mfree2
*

	include macro.h

start:
	move.l	A0,A1		* PSP-$10をA1に待避

	move.l	#BOTTOM-start+$F0,-(SP) * 自分に必要最小限のメモリ
	pea	16(A0)		* 現在のPSP
	FNC	$4A		* setblock
	addq.l	#8,SP

	pea	msg1
	FNC	$09		* print
	addq.l	#4,SP

	move.l	#$1000,-(SP)	* メモリブロック０を確保
	move.w	#0,-(SP)	* mallocと同様な割り当て方法
	FNC	$7D		* malloc2
	addq.l	#6,SP
	move.l	D0,mb0

	move.l	#0,D2
	bsr	area_disp

	move.l	#$1000,-(SP)	* メモリブロック１を確保
	move.w	#0,-(SP)	* mallocと同様な割り当て方法
	FNC	$7D		* malloc2
	addq.l	#6,SP
	move.l	D0,mb1

	move.l	#1,D2
	bsr	area_disp

	pea	msg2
	FNC	$09		* print
	addq.l	#4,SP

	move.l	mb0,-(SP)	* メモリブロック０を開放
	FNC	$7E		* mfree2
	addq.l	#4,SP

	move.l	mb1,D0
	move.l	#1,D2
	bsr	area_disp
	pea	msg3
	FNC	$09		* print
	addq.l	#4,SP
```

```
        move.l  #$1000,-(SP)     * メモリブロック2を確保
        move.w  #2,-(SP)         * メモリの上位から割り当て
        FNC     $7D              * malloc2
        addq.l  #6,SP
        move.l  D0,mb2

        move.l  mb1,D0
        move.l  #1,D2
        bsr     area_disp

        move.l  mb2,D0
        move.l  #2,D2
        bsr     area_disp

        pea     msg4
        FNC     $09              * print
        addq.l  #4,SP

        move.l  #$1000,-(SP)     * メモリブロック3を確保
        move.w  #0,-(SP)         * mallocと同様な割り当て方法
        FNC     $7D              * malloc2
        addq.l  #6,SP
        move.l  D0,mb3

        move.l  mb1,D0
        move.l  #1,D2
        bsr     area_disp

        move.l  mb2,D0
        move.l  #2,D2
        bsr     area_disp

        move.l  mb3,D0
        move.l  #3,D2
        bsr     area_disp

        pea     msg5
        FNC     $09              * print
        addq.l  #4,SP

        move.l  #$10,-(SP)       * メモリブロック4を確保
        move.w  #1,-(SP)         * 最小メモリ割り当て
        FNC     $7D              * malloc2
        addq.l  #6,SP
        move.l  D0,mb4
        move.l  mb1,D0
        move.l  #1,D2
        bsr     area_disp

        move.l  mb2,D0
        move.l  #2,D2
        bsr     area_disp

        move.l  mb3,D0
        move.l  #3,D2
        bsr     area_disp

        move.l  mb4,D0
        move.l  #4,D2
        bsr     area_disp

        move.l  #0,-(SP)
        FNC     $7E              * mfree2
        addq.l  #4,SP

        FNC     $00              * exit

area_disp:
        move.l  D0,A1
```

```
		pea	mb_msg
		FNC	$09		* print
		addq.l	#4,SP

		move.l	D2,D0
		bsr	hex2

		move.w	#':',-(SP)
		FNC	$02		* putchar
		addq.l	#2,SP

		move.l	A1,D0
		bsr	hex8

		move.w	#'-',-(SP)
		FNC	$02		* putchar
		addq.l	#2,SP

		move.l	-8(A1),D0
		bsr	hex8

		pea	crlf_msg
		FNC	$09		* print
		addq.l	#4,SP
		rts

hex8:					* １６進８桁表示サブルーチン
		move.l	D0,-(SP)
		swap	D0
		bsr	hex4
		move.l	(SP)+,D0
hex4:
		move.l	D0,-(SP)
		lsr.w	#8,D0
		bsr	hex2
		move.l	(SP)+,D0
hex2:
		move.l	D0,-(SP)
		lsr.w	#4,D0
		bsr	hex1
		move.l	(SP)+,D0
hex1:
		and.w	#$0F,D0
		add.w	#'0',D0
		cmp.w	#$3A,D0
		bcs	hex0
		add.w	#7,D0
hex0:
		move.w	D0,-(SP)
		FNC	$02		* putchar
		addq.l	#2,SP

		rts

msg1:
		dc.b	'★$1000バイトのメモリブロックを２つ(MB00,MB01)、従来と同じ
方法で取得',13,10
		dc.b	0
msg2:
		dc.b	'★MB00を開放',13,10
		dc.b	0
msg3:
		dc.b	'★$1000バイトのメモリブロック(MB02)をメモリの上位方向から取
得',13,10
		dc.b	0
msg4:
		dc.b	'★$1000バイトのメモリブロック(MB03)を従来と同じ方法で取得',
13,10
		dc.b	0
```

```
msg5:
		dc.b	'★$10バイトのメモリブロック(MB04)を最小メモリ割り当てで取得
',13,10
		dc.b	0
mb_msg:
		dc.b	'Memery Block ',0
crlf_msg:
		dc.b	13,10,0

		.even
mb0:
		ds.l	1
mb1:
		ds.l	1
mb2:
		ds.l	1
mb3:
		ds.l	1
mb4:
		ds.l	1

BOTTOM:		equ	*

		.end

*	3種類の方法を使ってメモリを取得してみせます。
*	表示を行うため、やや冗長なプログラムになっていますが、実行例と併せ
* て各メモリ取得方法の違いを確認してください。
*
*【実行例】
*	★$1000バイトのメモリブロックを２つ(MB00,MB01)、従来と同じ方法で取得
*	Memery Block 00:00172120-00173120
*	Memery Block 01:00173130-00174130
*	★MB00を開放
*	Memery Block 01:00173130-00174130
*	★$1000バイトのメモリブロック(MB02)をメモリの上位方向から取得
*	Memery Block 01:00173130-00174130
*	Memery Block 02:001AF000-001B0000
*	★$1000バイトのメモリブロック(MB03)を従来と同じ方法で取得
*	Memery Block 01:00173130-00174130
*	Memery Block 02:001AF000-001B0000
*	Memery Block 03:00172120-00173120
*	★$10バイトのメモリブロック(MB04)を最小メモリ割り当てで取得
*	Memery Block 01:00173130-00174130
*	Memery Block 02:001AF000-001B0000
*	Memery Block 03:00172120-00173120
*	Memery Block 04:00075550-00075560
*
```

SMPL 41

```
*
*	Function call sample program #41
*		$FFF3 diskred
*		$FFF4 diskwrt
*

		include macro.h

		.text
start:
		tst.b	(A2)+		* コマンドラインの文字数チェック
		beq	noname		*	0ならばnonameへ

		lea	arg1,A1		* argv[1]
		bsr	getarg		* arg取得サブルーチンコール
```

```
        tst.l   D0              * 文字数は０？
        beq     arg_err         *   ならばarg_errへ

        lea     arg2,A1         * argv[2]
        bsr     getarg          * arg取得サブルーチンコール
        tst.l   D0              * 文字数は０？
        beq     arg_err         *   ならばarg_errへ

        moveq   #0,D0
        move.b  arg2,D0         * サブドライブ名
        or.b    #$20,D0
        sub.w   #$60,D0         * ドライブ番号
        move.w  D0,subdrv

        moveq   #0,D0
        move.b  arg1,D0         * メインドライブ名
        or.b    #$20,D0
        sub.w   #$60,D0         * ドライブ番号
        move.w  D0,maindrv

        move.w  #1,-(SP)        * 1セクタ
        move.w  #0,-(SP)        * 第０セクタ
        move.w  maindrv,-(SP)   * メインドライブ番号
        pea     mainipl
        FNC     $F3             * diskred
        lea     10(SP),SP

        move.w  #1,-(SP)        * 1セクタ
        move.w  #0,-(SP)        * 第０セクタ
        move.w  subdrv,-(SP)    * サブドライブ番号
        pea     subipl
        FNC     $F3             * diskred
        lea     10(SP),SP

        lea     mainipl,A0

        lea     subipl,A1
        move.w  #1024-1,D0
chkloop:
        cmp.b   (A0)+,(A1)+     * IPLの内容をチェック
        bne     wrong           *   違っていたらwrongへ
        dbra    D0,chkloop

        pea     same_msg
        FNC     $09             * print
        addq.l  #4,SP

        FNC     $00             * exit

wrong:
        pea     wrong_msg
        FNC     $09             * print
        addq.l  #4,SP

        pea     inpptr
        move.w  #$0A,-(SP)      * key flush & gets
        FNC     $0C             * kflush
        addq.l  #6,SP

        pea     crlf_msg
        FNC     $09             * print
        addq.l  #4,SP

        move.b  inpptr+2,D0
        or.b    #$20,D0
        cmp.b   #'y',D0
        beq     iplcopy
        cmp.b   #'n',D0
        beq     nocopy
```

```
		bra	wrong

iplcopy:
		move.w	#1,-(SP)	* 1セクタ
		move.w	#0,-(SP)	* 第0セクタ
		move.w	subdrv,-(SP)	* サブドライブ番号
		pea	mainipl
		FNC	$F4		* diskwrt
		lea	10(SP),SP

		pea	copy_msg
		bra	disp_abort
nocopy:
		pea	nocopy_msg
		bra	disp_abort
noname:
		pea	noname_msg
		bra	disp_abort
arg_err:
		pea	arg_err_msg
disp_abort:
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00		* exit

getarg:					* arg取得サブルーチン(簡易型)
		moveq	#0,D0		* 文字数リセット
		moveq	#0,D7		* arg開始フラグリセット
getarg_loop:
		move.b	(A2)+,D1	* 1文字get
		beq	eol		*   $00ならばeolへブランチ
		cmp.b	#' ',D1		* スペース?
		beq	delim		*   ならばdelimへ
		cmp.b	#9,D1		* タブ?
		beq	delim		*   ならばdelimへ
					* それ以外ならば..
		moveq	#-1,D7		*   arg開始フラグをセット
		move.b	D1,(A1)+	* argバッファにコピー & ポインタ+1
		addq.l	#1,D0		* 文字数+1
		bra	getarg_loop
delim:
		tst.l	D7		* arg開始済み?
		beq	getarg_loop	*   まだならばgetarg_loopへ
eol:
		subq.l	#1,A2		* コマンドラインポインタ補正
		clr.b	(A1)+		* argにエンドマーク

		rts

		.data
same_msg:
		dc.b	'IPLセクタに相違は認められません',13,10
		dc.b	0
wrong_msg:
		dc.b	'IPLセクタの内容が違っています',13,10
		dc.b	'IPLセクタのコピーを行いますか? <Y/N> ',0
copy_msg:
		dc.b	'コピーを終了しました',13,10
		dc.b	0
nocopy_msg:
		dc.b	'コピーは行いません',13,10
		dc.b	0
noname_msg:
		dc.b	'ドライブネーム(MAIN,SUB)を指定してください',13,10
		dc.b	0
arg_err_msg:
```

```
                dc.b    'パラメータが異常です',13,10
                dc.b    0
crlf_msg:
                dc.b    13,10,0

inpptr:
                dc.b    2
                dc.b    0
                ds.b    2+1

                .bss
arg1:
                ds.b    40
arg2:
                ds.b    40
maindrv:
                ds.w    1
subdrv:
                ds.w    1
mainipl:
                ds.b    1024
subipl:
                ds.b    1024

                .end

*   コマンドラインで指定したドライブ(メイン、サブ)のＩＰＬセクタを比較
* し、違いが見付かった場合はメインのＩＰＬセクタをサブにコピーします。
* メイン側にはオリジナルのＨｕｍａｎのシステムディスクを用意することをお
* 勧めします。
*   ウイルスチェック等に利用できます。
*
* usage:        sample41 <main_drv> <sub_drv>
```

SMPL 42

```
*
*       Function call sample program #42
*               $FFF5 getindos
*

                include macro.h

                .text
TOP:            equ     *

process_name:
                dc.b    'SAMPLE42'
main:                                   * 割り込みで駆動される部分
                PUSH    D0-D4/A0-A1

                FNC     $F5             * getindos
                move.l  D0,A0           * InDOSフラグのアドレスをA0に
                tst.w   (A0)            * InDOSフラグは０か?
                beq     erase           *   ０ならばeraseへ
                move.b  #%1010,D1
                bra     disp
erase:
                move.b  #%0000,D1
disp:
                lea     indos_msg,A1
                move.w  #94,D2
                move.w  #31,D3
                move.w  #2-1,D4
                IOCS    $2F             * B_PUTMES (IOCS)
```

```
		POP	D0-D4/A0-A1

		rte

indos_msg:
		dc.b	'★',0

		.even

start:					* 初期化部分のスタート
		clr.l	-(SP)		* スーパーバイザモードへ
		FNC	$20		* super
		addq.l	#4,SP		* 返り値は読み捨てる
					* （ユーザモードへの復帰を省略したため）
		move.w	#$4D,-(SP)	* INTベクター#$46 : V-DISP割り込み(TIMER-A)
		FNC	$35		* intvcg
		addq.l	#2,SP

		move.l	D0,A1
		lea	-8(A1),A1	* process_name
		cmp.l	#'SAMP',(A1)	* process_nameの前４バイトは一致するか？
		bne	not_exist	*   一致しなければnot_existへ
		cmp.l	#'LE42',4(A1)	* process_nameの後ろ４バイトは一致するか？
		bne	not_exist	*   一致しなければnot_existへ
exist:					* すでにSAMPLE42が組み込まれていた場合
		move.w	#$000,D1	* dammy
		clr.l	A1		* 割り込み解除
		IOCS	$6C		* VDISPST (IOCS)

		move.l	A1,-(SP)	* 常駐していたSAMPLE46のPSP+$10
		FNC	$49		* mfree
		addq.l	#4,SP

		pea	free_msg
		FNC	$09		* print
		addq.l	#4,SP

		FNC	$00		* exit

not_exist:
		move.w	#$101,D1	* 垂直表示期間１回毎に割り込み
		lea	main,A1		* エントリーはmain
		IOCS	$6C		* VDISPST (IOCS)

		pea	keep_msg
		FNC	$09		* print
		addq.l	#4,SP

		move.w	#0,-(SP)	* 終了コード０
		move.l	#start-TOP,-(SP)	* 常駐させるバイト数
		FNC	$31		* keeppr


free_msg:
		dc.b	'常駐解除しました',13,10
		dc.b	0
keep_msg:
		dc.b	'常駐します',13,10
		dc.b	0

		.end	start

*  約１／６０秒ごとにＨｕｍａｎのファンクションコールのＩｎＤＯＳフラ
* グを調べ、ＤＯＳファンクションコール実行中であれば画面右下に★印を表
* 示する常駐プログラムです。
*  一度実行すると常駐し、もう一度実行すると常駐解除します。
```

SMPL 43

```
*
*       Function call sample program #43
*               $FFF6 farcall
*

                include macro.h

                .text
start:
                pea     come_here       * come_hereをコール
                FNC     $F6             * farcall
                addq.l  #4,SP

                move.l  work,D0

                bsr     hex8            * １６進８桁表示ルーチンコール

                pea     crlf_msg
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00

come_here:                              * ここに飛んでくる
                move.l  0,work          * スーパーバイザエリア(ここでは$0)を読んでみ
る
                rts

hex8:                                   * １６進８桁表示サブルーチン
                move.l  D0,-(SP)
                swap    D0
                bsr     hex4
                move.l  (SP)+,D0
hex4:
                move.l  D0,-(SP)
                lsr.w   #8,D0
                bsr     hex2
                move.l  (SP)+,D0
hex2:
                move.l  D0,-(SP)
                lsr.w   #4,D0
                bsr     hex1
                move.l  (SP)+,D0
hex1:
                and.w   #$0F,D0
                add.w   #'0',D0
                cmp.w   #$3A,D0
                bcs     hex0
                add.w   #7,D0
hex0:
                move.w  D0,-(SP)
                FNC     $02             * putchar
                addq.l  #2,SP

                rts

                .data
crlf_msg:
                dc.b    13,10,0

                .bss
work:
                ds.l    1

                .end
```

```
*  プログラム内のcome_hereを＄ＦＦＦ６farcallで呼んでみます。farcallで
* 呼ばれた先ではスーパーバイザモードになっているので、スーパーバイザ領
* 域もアクセスすることができます。
```

SMPL 44

```
*
*       Function call sample program #44
*               $FFF7 memcpy
*

                include macro.h

                .text
start:
                tst.b   (A2)+           * コマンドラインの文字数チェック
                beq     noadr

                moveq   #0,D7           * コマンドライン→ロングワードデータ
loop:
                move.b  (A2)+,D0
                beq     eol
                sub.b   #'0',D0
                bcs     illegal
                and.b   #%11011111,D0
                cmp.b   #$10,D0
                bcs     shift
                sub.b   #7,D0
shift:
                cmp.b   #$10,D0
                bcc     illegal
                lsl.l   #4,D7
                or.b    D0,D7
                bra     loop
eol:
                move.w  #2,-(SP)        * word
                pea     work
                move.l  D7,-(SP)
                FNC     $F7             * memcpy
                lea     10(SP),SP
                tst.l   D0
                bmi     adr_err

                move.w  work,D0
                bsr     hex4            * １６進４桁表示ルーチンコール

                pea     crlf_msg
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit

noadr:
                pea     noadr_msg
                bra     disp_abort
illegal:
                pea     illegal_msg
                bra     disp_abort
adr_err:
                pea     adr_err_msg
disp_abort:
                FNC     $09             * print
                addq.l  #4,SP

                FNC     $00             * exit
```

```
hex4:                                   * １６進４桁表示ルーチン
                move.l  D0,-(SP)
                lsr.w   #8,D0
                bsr     hex2
                move.l  (SP)+,D0
hex2:
                move.l  D0,-(SP)
                lsr.w   #4,D0
                bsr     hex1
                move.l  (SP)+,D0
hex1:
                and.w   #$0F,D0
                add.w   #'0',D0
                cmp.w   #$3A,D0
                bcs     hex0
                add.w   #7,D0
hex0:
                move.w  D0,-(SP)
                FNC     $02             * putchar
                addq.l  #2,SP

                rts

                .data
noadr_msg:
                dc.b    'アドレスを指定してください',13,10
                dc.b    0
illegal_msg:
                dc.b    'アドレスの表現が異常です',13,10
                dc.b    0
adr_err_msg:
                dc.b    'アドレスエラーです',13,10
                dc.b    0
crlf_msg:
                dc.b    13,10,0

                .bss
work:
                ds.w    1

               .end

*    コマンドラインで指定したアドレスのワードデータを表示します。
*
* usage:        sample44 <address>
```

ASKSMPL.S

```
*
*       FP CALL sample program
*               $FF22 knjctrl
*

                include macro.h

                .text
start:
                tst.b   (A2)+               * コマンドラインの文字数チェック
                beq     noname              *   0ならnonameへ

                move.w  #0,-(SP)            * READ
                move.l  A2,-(SP)            * コマンドライン＝ファイルネーム
                FNC     $3D                 * open
                addq.l  #6,SP
                tst.l   D0                  * エラー？
                bmi     open_err            *   ならばopen_errへ

                move.w  D0,handl            * ファイルハンドルをhandlへ保存

                move.l  #0,-(SP)            * かな漢字変換モードロック
                move.l  #7,-(SP)            * FP CALL #7
                FNC     $22                 * knjctrl
                addq.l  #8,SP

                move.l  #0,-(SP)            * 通常モード
                move.l  #1,-(SP)            * FP CALL #1
                FNC     $22                 * knjctrl
                addq.l  #8,SP
                clr.b   eof_flag            * ファイル終端フラグクリア
line_start:
                clr.w   counter             * カウンタクリア
                clr.b   onemore_flag        * 『もう一文字』フラグクリア
                clr.b   lf_flag             * LFフラグクリア
                lea     inbuf,A1            * 入力バッファのポインタ
line_loop:
                move.w  handl,-(SP)
                FNC     $1B                 * fgetc
                addq.l  #2,SP
                tst.l   D0                  * ファイル終端ならば
                bmi     eof                 *   eofへブランチ

                move.b  D0,(A1)+            * バッファへ
                addq.w  #1,counter          * 入力カウンタ＋1

                cmp.b   #$20,D0             * 入力された文字が$20以下なら
                bcs     ctrl                *   ctrlへブランチ
                cmp.b   #$80,D0             * 入力された文字が$80以下なら
                bcs     ank                 *   ankへブランチ
                cmp.b   #$A0,D0             * 入力された文字が$A0以下なら
                bcs     sjis                *   sjisへブランチ
                cmp.b   #$E0,D0             * 入力された文字が$E0以上なら
                bcs     ank                 *   ankへブランチ
sjis:
                tst.b   onemore_flag        * 『もう一文字フラグ』が立っていたら
                bne     ank                 *   ankへ
                move.b  #-1,onemore_flag * 『もう一文字フラグ』を立てる
                bra     line_loop

ank:
                cmp.w   #78,counter         * 入力文字数は７８バイト以下？
                bcs     line_loop           *   ならばline_loopへ
```

```
                clr.b   (A1)            * 行末コード付加
                bra     henkan

eof:
                move.b  #-1,eof_flag    * ファイル終端フラグを立てる
                clr.b   (A1)            * 行末コード付加
                bra     henkan

ctrl:
                clr.b   -1(A1)          * バッファの最後を行末コードに
                cmp.b   #$0A,D0         * 入力された文字はLF?
                bne     henkan          *   でなければhenlanへ
                move.b  #-1,lf_flag     * LFフラグ立てる

henkan:
                pea     kouzoku         * このプログラムでは利用しない
                pea     kouho           * このプログラムでは利用しない
                pea     inbuf           * 入力バッファを変換
                move.l  #19,-(SP)       * FP CALL #19
                FNC     $22             * knjctrl
                lea     16(SP),SP

                pea     retbuf          * 変換後の文字列が入るバッファ
                move.l  #24,-(SP)       * FP CALL #24
                FNC     $22             * knjctrl
                addq.l  #8,SP
                tst.l   D0              * エラー?
                bne     henkan_err      *   ならばhenkan_errへ

                pea     retbuf
                FNC     $09             * print
                addq.l  #4,SP

                tst.b   lf_flag         * LFフラグが立っている?
                beq     lbl1            *   立っていなければlbl1へ

                pea     crlf_msg
                FNC     $09             * print
                addq.l  #4,SP
                clr.b   lf_flag         * LFフラグクリア
lbl1:
                tst.b   eof_flag        * ファイル終端?
                beq     line_start      *   でなければline_startへ

                move.w  handl,-(SP)
                FNC     $3E             * close
                addq.l  #2,SP

                bra     exit

noname:
                pea     noname_msg
                bra     disp_abort
open_err:
                pea     open_err_msg
                bra     disp_abort
henkan_err:
                pea     henkan_err_msg
disp_abort:
                FNC     $09             * print
                addq.l  #4,SP
exit:
                move.l  #1,-(SP)        * 変換モードアンロック
                move.l  #7,-(SP)        * FP CALL #7
                FNC     $22             * knjctrl
                addq.l  #8,SP

                FNC     $00             * exit
```

```
		.data
noname_msg:
		dc.b	'ファイル名を指定してください',13,10
		dc.b	0
open_err_msg:
		dc.b	'ファイルがみつかりません',13,10
		dc.b	0
henkan_err_msg:
		dc.b	'変換中にエラーが発生しました',13,10
		dc.b	0
crlf_msg:
		dc.b	13,10,0

		.bss
handl:
		ds.w	1
inbuf:
		ds.b	80
retbuf:
		ds.b	80
kouzoku:
		ds.b	80
kouho:
		ds.b	80

counter:
		ds.w	1
eof_flag:
		ds.b	1
onemore_flag:
		ds.b	1
lf_flag:
		ds.b	1


		.end

*	ＦＰコールを利用して、コマンドラインで指定したファイルの内容を漢字
* 変換しつつ表示します。候補の選択や文節の切り直しなどは一切できません。
* フロントプロセッサ自身の変換性能が問われるプログラムと言えましょう。
*	なお、途中でCTRL-Cした場合、かな漢字変換モードがロックされた状態で
* 終了してしまいますのでご注意ください。
*
* usage:		asksample <filename>
```

第2部

IOCSコール

1章 IOCSを使うための予備知識

X68000 のハードウェア関係のサービス・ルーチン群「IOCS コール」について，基本的な点を説明します．

1.1 IOCSとは?

X68000は標準でいくつかの高度な外部デバイスを備えています．そのため，これらを活用した高度な処理が可能です．

ところが，実際にこれらの高度なハードウェアを活用するプログラムを作る場合，パラメータの数が多くなったり，割り込み処理によって制御を行なう必要があったりするため、かなり難しいプログラムになります．量的にみてもデバイスの数が少なくないため，それぞれのデバイスに対応した基本的サブルーチンを独自に作成する必要があり，たくさんのプログラムを作ることになるため大変です．また，同じような処理をする場合は，サブルーチンを共有した方が合理的です．

さらに，安全性の面からみても個々に処理ルーチンを作るのは問題があります．これは，68000CPUがシステム保護機能として「スーパーバイザ・モード」と「ユーザー・モード」の２つのモードを持っているためです．X68000はハードウェア関係のI/Oポートは安全のためすべてスーパーバイザ・エリアにマッピングされており，ユーザー・モードではI/Oを直接制御できません．このため，I/Oを直接制御するプログラムはスーパーバイザ・モードで動作させることになり，プログラム・ミスがあった場合暴走してしまいます．

これらの問題を解決するため，X68000にはROMの形で基本的なハードウェア制御をするサブルーチン群が用意されています．それが，これから説明していく「IOCSコール」です．

IOCSコールはX68000のハードウェアのほとんどを制御でき，レジスタやメモリに入力値を設定して呼び出すだけでハードウェアを制御できます．また，コール時にはTRAP命令を使うためユーザーモードからのアクセスが可能です．よって，IOCSコールを使えば，安全性の高いユーザー・モードで簡単にハードウェアを制御できます．

このようにIOCSコールを使うことによって楽にプログラムを作ることができるわけですが，問題点もあります．それは，「処理速度が遅い」ということです．IOCSコールはさまざまなプログラムからコールされるため，非常に汎用性が高くなっています．また，同時に多機能である必要があるため，レジスタなどで指定する項目が多くなっています．そのため，これらの点が原因となってIOCSコールの処理速度はかなり遅くなっています．IOCSを使うプログラムを作る場合は，このことを考えに入れておく必要があります（とくに，グラフィック関係)．

1.2 IOCS コールの使用法とコール時のプロセスについて

IOCSコールは基本的には次のようにコールします．

```
move. l   #IOCS ナンバー, D0
trap      #15
```

IOCSナンバーは０から$FFまでの数で，呼び出すサブルーチンを指定します．

IOCSによっては入力値をD0.l以外のレジスタやメモリに設定しておく必要があります．また，返り値があるものはリターンしてきた時点でレジスタやメモリに結果が格納されています．どのIOCSコールでも，D0.lはつねに破壊されます（または，結果が格納される)．

68000CPUはTRAP命令に出会うと，スーパーバイザ・モードになり，PCとSRをスーパーバイザ・スタックに待避させ，ベクタナンバーで示されるアドレスの内容が示すアドレスにジャンプします．X68000の場合はジャンプ先にIOCS処理ルーチンがあり，D0.bで指定される機能を実行します．そして，処理が終わると結果を格納して例外処理からのリターンをします（RTE)．このときにPCとSRが元に戻り，元のモードになった後，TRAP命令以降のプログラムを実行します．

2章 IOCSを使うためのハード基礎知識

X68000のハードは非常に高度であるため，IOCSコールが用意されていてもハードについてわからない点があると制御できません．そこで，この章ではIOCSコールによってハードを制御するために必要な説明をしていきます．

2.1 キー入力関係

1. X68000のキー入力について

X68000はサブCPU (80C51) がキーボードを監視し，メインCPUにデータを送っています．また，LEDモードなどのデータをメインCPUから受け取っています．

キーが押された場合，サブCPUはメインCPUに対して割り込みをかけ，押されたキーの情報を送ります．メインCPUは割り込みサブルーチン内でサブCPUのデータを受け取り，必要ならばハードコピーなどのリアルタイム動作を行なったのち，キーバッファへデータを書き込みます．このバッファを読み出すにはIOCS $00を使います．また，IOCS $02, $04用テーブルを書き換えます．入力されるキーによって入力モードが変わる場合，LEDモードを変更します．

キー・リピートの入力はキーが押された場合と同じ作業を行ないます．キーが離された場合はIOCS $02, $04用のテーブルを書き換え，特殊なキー (SHIFT, CTRL, OPT.1, OPT.2) はキーバッファへデータを書き込みます．

このように，IOCSコールのキー入力は割り込みで得たデータを読み出すだけであるため，IOCSコール用キー入力割り込みルーチンが動作しない場合はデータが更新されなくなります．また，リアルタイム処理も行なわれなくなります．

2. ソフトウェア・キーボード・電卓について

ソフトウェア・キーボードはIOCSのレベルでサポートされている仮想キーボードです．IOCSコールの範囲では，実際のキーボードと同じものとして扱われています．

電卓による入力もIOCSのレベルでサポートされています．そのため電卓モードにはいると，IOCS

$00	キー入力バッファ・データ読み出し
$01	キー入力バッファ・データを調べる
$02	シフトキー状態を調べる
$03	キー入力関係の初期化 (キー入力バッファをクリア)
$04	キー入力状態をキーごとに調べる
$05	キー入力発生
$06	キーボードLEDモードの設定
$07	キーボードLEDモードをキー入力モードに合わせる
$08	キー・リピートの開始時間設定
$09	キー・リピート間隔の設定
$0A	OPT.2によるテレビコントロール許可
$0B	OPT.2によるテレビコントロール禁止
$0D	キーボードLEDモード&キー入力モードを設定

$00, $01の出力に影響が出ます（電卓入力になるキー入力がIOCS $00, $01には現れなくなります）．また，電卓モードにはいるためのキー操作は (OPT.1＋OPT.2の操作) IOCS $00, $01に出力されます．

3. LEDモードと入力モード

LED付きのキー(ひらがな，全角，かな，ローマ字，コード入力，CAPS) のLEDの状態が「LEDモード」，入力されたキーに加えられる修正の種類の状態が「入力モード」です．通常はこの2つのモードは同じ状態になっていますが，IOCSコールの中には片方のモードしか変更しない物があります．このようなIOCSコールを使った場合，LEDの状態と入力モードが一致しなくなる場合があります．

★グラフィック画面としてのIOCS	
$10	CRT モード設定
$11	コントラスト設定
$12	HSV データから RGB データを計算
$13	テキスト・パレット定義
$14	テキスト・パレット独立定義
$15	アクセス・プレーン設定
$17	テキスト VRAM からのバイト単位読み込み
$18	テキスト VRAM へのバイト単位書き込み
$1A	テキスト VRAM からのドット単位読み込み
$1B	テキスト VRAM へのドット単位書き込み
$1C	テキスト VRAM へのドット単位書き込み（クリッピング付き）
$1D	表示位置設定
$93	テキスト画面の表示オン・オフ
$D3	水平線を描く
$D4	垂直線を描く
$D6	ボックスを描く
$D7	塗り潰しボックスを描く
$D8	指定範囲を反転する
$DF	ラスターコピーによる VRAM データ・コピー
★キャラクタ画面としてのIOCS	
$1E	カーソル一時停止解除
$1F	カーソル一時停止
$20	文字表示
$21	文字列表示
$22	文字属性の設定（カラー）
$23	カーソル位置指定
$24	カーソルを１行下へ移動する（スクロールする位置ならばスクロールする）
$25	カーソルを１行上へ移動する（スクロールする位置ならばスクロールする）
$26	カーソルを指定行数上へ移動する（スクロールなし）
$27	カーソルを指定行数下へ移動する（スクロールなし）
$28	カーソルを指定桁数右へ移動する（スクロールなし）
$29	カーソルを指定桁数左へ移動する（スクロールなし）
$2A	複数行消去（スクロールなし）
$2B	複数桁消去
$2C	複数行挿入（スクロールする）
$2D	複数行消去（スクロールする）
$2E	表示範囲指定
$2F	画面の絶対位置に文字列を表示する（コンソール画面範囲の影響を受けない）
$AE	カーソル表示
$AF	カーソル非表示

1.グラフィック画面としてのテキスト画面

★テキスト画面の構成

X68000のテキスト画面はIOCSコールによってテキスト文字を表示させるための画面ですが，実際には「1024×1024ドット，ドット単位に65536色中の16色が表示可能」というグラフィック画面です．構成はP.303 の図のようになっています．

IOCSでは文字表示用にプレーン０＆１を使い，ソフトウェアキーボード・電卓・マウス・カーソルの表示用にプレーン２＆３を使っています．そして，テキスト・パレットの４から７までと，８から$Fまでにそれぞれ同じカラーコードを定義することによって，ソフトウェア・キーボードなどが，プレーン０＆１に表示されている文字による影響を受けないようにしています．このことをサポートしているパレット定義IOCSが$13，サポートしていないIOCSが$14です．また，実画面は1024×1024ドット分ありますが，表示できる範囲は最大で768×512ドットです（24KHzモードを使わない場合）．表示画面は，実画面内に収まる範囲において，任意の位置に設定できます．

テキスト・パレットは，スプライト・パレット・ブロック０と同じものです．

ラスタ・コピーはCRTCの機能で，指定したラスタ（X方向ラインのことで，縦４ドットごとに制御できる）のVRAM内容を他のラスタの位置へコピーする機能です（高速にコピーできる）．

2．キャラクタ画面としてのテキスト画面

X68000のIOCSは，1024×1024ドットのテキスト画面のうち指定した範囲にコンソール画面を設定できます．そして，その範囲内でのカーソル移動，文字表示などをサポートしています．通常の文字表示で扱う座標はすべてコンソール画面内での相対座標です．

IOCS $20, $21では1バイト・コントロールコードが使えます．また，2バイト文字は1バイトごとに送っても表示されます．

通常の文字表示のときには基本的にコンソール画面以外には影響は出ませんが，スクロールする場合はコンソール画面がある横方向のラインも一緒にスクロールしてしまいます．

色指定は“1．グラフィック画面としてのテキスト画面”で述べたように，文字表示用に2つのプレーンしか割り当てられていないため，4色しか指定できません．ただし，表示方法が4種類あります（通常，強調，通常反転，強調反転）．

2.3 グラフィック画面

$10	CRTモード設定
$1D	表示位置設定
$90	グラフィック表示モードにする
$91	グラフィック画面モードの設定
$92	グラフィック・ページ間・テキスト&スプライト画面間のプライオリティ設定
$93	表示切り替え，特殊モード設定
$94	グラフィック・パレット設定
$95	書き込みパレットコード設定（$9A，$9B，$9C用）
$96	読み出し，書き込みを行なうグラフィック・ページを設定する
$97	グラフィック画面からデータを読み出す
$98	グラフィック画面にデータを書き込む（透明パレットコード設定可能）
$99	グラフィック画面にデータを書き込む
$9A	グラフィック画面にビット・パーターンを書に込む（ORをとる）
$9B	グラフィック画面にビット・パターンを書き込む
$9C	グラフィック画面にビット・パターンを拡大書き込み（ORをとる）
$B1	読み出し，書き込みページの設定を行なう
$B2	表示ページ設定
$B3	表示位置設定
$B4	ウィンドウを設定する（IOCS$BO番台に有効）
$B5	グラフィック画面クリア（WIPE）
$B6	ドットセット（PSET）
$B7	指定座標のパレットコードを調べる（POINT）
$B8	直線を描く（LINE）
$B9	ボックスを描く（BOX）
$BA	塗り潰しボックスを描く（FILL）
$BB	円を描く（CIRCLE）
$BC	塗り潰す（PAINT）
$BD	文字を表示する（SYMBOL）
$BE	グラフィック画面からデータを読み出す（GET）
$BF	グラフィック画面にデータを書き込む（PUT）

1．グラフィック画面の概要

X68000はグラフィック表示用に512KBのグラフィックVRAMを装備しています．そして，4種類の画面モードを取ることが可能になっています．画面モードは次のとおりです．

1024×1024ドット×1枚
　ドット単位に65536色中の16色を表示可能
512×512ドット×1枚
　ドット単位に65536色を表示可能
512×512ドット×2枚
　ドット単位に65536色中の256色を表示可能
512×512ドット×4枚
　ドット単位に65536色中の16色を表示可能

IOCSではグラフィック画面に対するモード設定，表示位置指定，書き込み，読み出し，特殊モードの設定をサポートしています．

モード設定はIOCS $10で行ないます．ただし，IOCS $10だけではグラフィック画面が表示モード

になりません．IOCS $90で表示モードにしてください．表示モードになると，各グラフィック画面が表示のオンオフ，プライオリティ，パレット・コードなどによって指定される形式で表示されます．

2．グラフィック画面の機能

グラフィック画面はさまざまな特殊機能を備えています．IOCSではそのほとんどをサポートしています．

<表示位置>

各グラフィック・ページを任意の位置から表示させることができます．X方向，Y方向について，グラフィック画面の大きさの値まで指定できます(球面スクロール)．

<プライオリティ>

各グラフィック・ページ間で表示優先順位を設定できます．パレットコードが0のとき，その点が透明になります．また，グラフィック，テキスト，スプライトの各画面間でのプライオリティも設定できます．

<半透明機能>

半透明機能は複数の画面のカラーコードを合成して表示する機能です．カラーコードの合成は，それぞれのコードをR(赤)，G(緑)，B(青)，I(輝度)に分解して，輝度以外の平均を取ることによって行ないます．合成後の輝度は，半透明エリアを指定していない側のデータがそのまま使用されます．合成パターンは，つぎの4つがあります．

①グラフィック画面と
テキスト・パレットコード0との間

グラフィック画面全体にテキスト・パレットコード0のカラーコードを半透明合成させます．

②グラフィック・ページと
テキスト&スプライト画面との間

グラフィック・ページよって指定されたエリアで，グラフィック画面とテキスト&スプライト画面との間の半透明合成を行ないます．このとき，半透明エリアを指定するグラフィック・ページは，全表示画面中で最も表示優先順位が高くなければなりません．

③グラフィック・ページ間

全表示画面中で最も表示優先順位が高いグラフィック・ページと，次に優先順位が高いグラフィック・ページとの間で半透明合成を行ないます．半透明エリアは最も優先順位が高いグラフィック・ページで指定します．

④グラフィック画面と
テレビ&ビデオ画面との間

全表示画面中で最も表示優先順位が高いグラフィック・ページと，テレビ&ビデオ画面との間で半透明合成を行ないます．半透明エリアは最も優先順位が高いグラフィック・ページで指定します．

これらの合成パターンは重複できる組み合わせがあります．また，半透明エリア指定を行なうために，グラフィックで使える色数が半分になります．これは，パレットコードの最下位ビットが半透明を行なうか行なわないかを示すフラグの意味を持つためです．よってエリア指定を行なうページでは，奇数のパレットコードは無効となり半透明を受けるページではエリア内のカラーコードは奇数パレットの内容が，エリア外のカラーコードは偶数パレットの内容が，それぞれ使われます．

<特殊プライオリティ機能>

特殊プライオリティ機能とは，グラフィック画面の優先順位がテキスト・スプライト画面より低い場合，指定したエリアだけはグラフィックの優先順位を他の画面より高くするという機能です．

エリアの指定はグラフィック画面中最も表示優先順位が高いグラフィック・ページで行ないます．この指定を行なうために，グラフィックで使える色数が半分になります．これは，パレットコードの最下位ビットが特殊プライオリティを行なうか行なわないかを示すフラグの意味を持つためです．よって奇数のパレットコードは無効となります．

<半透明，特殊プライオリティエリア指定>

半透明・特殊プライオリティを行なう場合，機能が働くエリアを指定する場合があります．この指定はグラフィック画面中最も表示優先順位が高

のグラフィック・ページのデータの最下位ビットによって行ないます(1のとき指定)．よって，データが示すパレットコードは偶数のものだけになるので，パレットコードは半分しか使えなくなります（データが奇数→最下位ビットを0にしたときに示されるパレットコードを半透明・特殊プライオリティ表示）．

2.4 ディスク・ドライブ

1．2HDディスク・ドライブ

X68000のフロッピーディスク・ドライブは，5インチ両面高密度タイプ（2HD）となっています．このディスクはディスクの片面につき77本のトラックがあり，アンフォーマット時で1.6MBの容量があります．

実際に使う場合はフォーマットを行ない，セクタを作ってデータが書き込めるようにします．セクタの順番は，トラック0のサイド0の1番セクタが最初のセクタです．それ以降は，そのトラックのサイドにある最後のセクタまでセクタ番号順に並び，その次がサイド1の同じトラックのセクタとなっています．この後はトラック・ナンバーが1つ増えます．

フォーマットする場合はIOCS $4Dを使います．セクタの大きさは128，256，512，1024バイトが使えます．代表的なフォーマット・パターンとしてはHuman68Kの1024バイト・セクタ×8，XOS－9の256バイト・セクタ×26などがあります．通常の読み出しはIOCS $46，書き込みはIOCS $45を使います．

X68000本体に装備されているドライブは，ディスクのセット・イジェクトが自動的にできるようになっています．また，IOCS $4E，$4Fによって，ソフトウェアによるイジェクト禁止が可能です．

ディスクは他のハードと違ってエラーが起こる可能性があります．データ・エラーチェックはCRC巡回符号によって，ハードウェアで行なわれます．そして，エラーが検出されるとCPUの方にエラーの種類，発生位置などの情報を送ります．

$40	指定位置へのシーク
$41	指定データとディスクの内容を比較（ベリファイ）
$42	診断のための読み込み（通常は使わない）
$43	ディスク関係の初期化
$44	ディスクステータスの調査
$45	書き込み
$46	読み込み
$47	トラック0へのシーク，ドライブの存在を調べる
$48	ハードディクスの代替トラックの設定
$49	破損データの書き込み（通常は使わない）
$4A	IDデータ読み込み
$4B	ハードディスクの破壊トラックを使用不能にする
$4C	破損データの読み込み（通常は使わない）
$4D	フォーマット
$4E	ドライブの調査，イジェクト許可・禁止
$4F	強制イジェクト

2．ハードディスク・ドライブ

X68000のIOCSはハードディスク関係の処理をサポートしています．しかし，ハードディスクはフロッピーディスクと違って容量が大きいため，破壊してしまったときの復旧が難しく，同時に非常に面倒です．ですから，ユーザーがハードディスクを直接アクセスするということはあまり良いことではありません．ハードディスクはDOSなどを通してアクセスするようにしてください．

個々のIOCSの説明のところでは，IOCSの入出力条件と実行内容のみを示すに止めてあります．どうしても使う場合は，それなりの覚悟を決めて使ってください．

2.5 ADPCM

X68000は「ADPCM」方式のサンプリング入出力が可能です.「ADPCM」方式は, サンプリングしたデータを1つ前のデータと比較して, その差分をPCMデータとする方法です. X68000の場合, サンプリング周波数は, 3.9KHzから15.6KHzまでの5種類があります. また, サンプリング入力は8ビット(0～255)で量子化を行ない, そのデータと1つ前のデータの差分を4ビットで(符号ビットも含む)I/Oに出力します. サンプリング出力は, I/Oから4ビット差分データを受け取り, サンプリング入力のときと逆に合成して音声出力します. データ量は, 3.9KHzの場合毎秒1950バイト, 15.6KHzの場合毎秒7800バイトとなります.

IOCSでは$60～$67がPCM用に割り当てられています. 入出力形式は, 単純な転送($60, $61), チェーン・テーブルを使う複数データ転送($62, $63), アレイチェーン・テーブルを使う複数データ転送($64, $65), の3通りがあります(DMAを使うため). データ量が少ない場合はすぐにリターンしてきます(出力終了まで待たない).

$60	ADPCMへデータを出力
$61	ADPCMからデータを入力
$62	ADPCMへデータを出力(チェーン・テーブルによる複数データ出力)
$63	ADPCMからデータを入力(チェーン・テーブルによる複数データ入力)
$64	ADPCMへデータを出力(アレイ・チェーン・テーブルによる複数データ出力)
$65	ADPCMからデータを入力(アレイ・チェーン・テーブルによる複数データ入力)
$66	ADPCMの実行モードを調べる
$67	ADPCMの実行を制御する

2.6 FM音源OPMについて

X68000はFM音源用にOPMを搭載しています. 8音までの同時発音ができるため, ステレオ効果を出すことができます. また, 低周波発振器(LFO)による周波数変調, 振幅変調がかけられます.

発音させるには, 音色データを各レジスタにセットした後, キー・オンにします.

OPMの割り込みを使う場合はIOCS $6AでMFPに対する設定を行ない, また, OPMのレジスタも設定します.

$68	FM音源レジスタへデータを書き込む
$69	FM音源の状態を調べる
$6A	FM音源割り込みの設定

$01 テスト	通常bit 1以外は使いません。bit 1に1を書き込み, その後0を書き込むと, LFOの出力する波形がスタート位置に戻ります.
$08 ノートオン･オフ	bit 2～0にチャンネルナンバーを, bit 6～3にスロットのオン･オフ(bit 6＝C 2　5＝M 2　4＝C 1　3＝M 1のキーオン・オフ　1ときオン)を書き込むと, そのチャンネルのスロットが指定した動作をします.
$0F ノイズ	bit 7を1にするとチャンネル8のC2がノイズになります(ホワイトノイズ). bit 4～0でノイズの周波数を指定します.
$10 タイマーA $11 タイマーA	$10の内容×4＋($11の下位2ビットの内容)をXとすると, (1024－X)×64÷4000000(ms)の間隔でオーバーフローを発生します.
$12 タイマーB	$12の内容をXとすると, (256－X)×1024÷4000000(ms)の間

	隔でオーバーフローを発生します．
$14　タイマー制御	bit 0を1にするとタイマーAが，bit 1を1にするとタイマーBがカウントを始めます． bit 2を1にするとタイマーAオーバーフロー時に割り込みがかかります． bit 3を1にするとタイマーBオーバーフロー時に割り込みがかかります． bit 4を1にするとタイマーAオーバーフローフラグをリセットします． bit 5を1にするとタイマーBオーバーフローフラグをリセットします． bit 7を1にするとタイマーAオーバーフロー時にすべてのスロットをキー・オンします． いちど割り込みがかかった場合，オーバーフロー・フラグをリセットしないと，それ以後割り込みがかかりません．どちらのタイマーから割り込みがかかったかはIOCS$69で調べます．
$18　LFO周波数	LFO周波数を指定します．ここで指定した値と実際の周波数は比例していません．
$19　AMD・PMD設定	bit 7を1にすると，書き込んだときのbit 6～0がPMDの値となります． bit 7を0にすると，書き込んだときのbit 6～0がPMDの値となります．
$1B ウェーブフォーム	bit 1～0でLFOのウェーブ・フォームを指定します．bit 7～6は0にします．
$20～$27 チャンネルごとの指定	bit 7～6　LR bit 5～3　フィードバックレベル bit 2～0　アルゴリズム
$28～$2F チャンネルごとの指定	bit 6～4　オクターブ bit 3～0　符号（定義されていない数値があります） $0　D#　　$6　G# $1　E　　$8　A $2　F　　$9　A# $4　F#　　$A　B $5　G　　$C　C $D　C# $E　D
$30～37 チャンネルごとの指定	bit 7～2　細かい音階指定（符号の間の音程を指定できる） $28～$2Fで指定した符号が表わす音程と，次の音程（1度上）との間を64段間に分けて指定します．
$38～$3F チャンネルごとの指定	bit 6～4　PMS bit 1～0　AMS
$40～$5F スロットごとの指定	チャンネル1はスロット1，9，17，25で構成される． bit 6～4　DT1 bit 3～0　MUL

$60〜$7F スロットごとの指定	bit 6〜0　TL
$80〜$9F スロットごとの指定	bit 7〜6　KS bit 4〜0　AR
$A0〜$BF スロットごとの指定	bit 7　AMS イネーブル bit 4〜0　D1R
$CO 〜$DF スロットごとの指定	bit 7〜6　DT2 bit 4〜0　D2R
$E0〜$FF スロットごとの指定	bit 7〜4　DIL bit 3〜0　RR

2.7 各種タイマー

X68000は割り込みコントローラによって割り込みの許可・禁止などを行なっています．ただし，IOCSではすべての割り込みはサポートしていません（例：キーボード割り込み）．

割り込み発生時に処理アドレスにジャンプする時点で，CPUはスーパーバイザ・モードになっています．割り込み処理を行なう場合，割り込みルーチンの先頭でレジスタを保存し，終了するときに復帰させます．割り込みからのリターンは「RTE」を使います．なお，割り込みをスタートさせる前に，処理ルーチンを用意してください．

$6A	OPM による割り込み
$6B	タイマー D による割り込み
$6C	垂直同期による割り込み
$6D	ラスター操作による割り込み
$6E	水平同期による割り込み
$6F	プリンタによる割り込み

これらの割り込みはほとんどが一定周期で割り込みを発生するタイプですが，プリンタ割り込みはプリンタがBUZYでなくなったときに割り込みがかかります．

2.8 マウス関係

X68000には標準でマウスがついています．IOCSではマウスとデータをやりとりするためのサブルーチンをサポートしています．IOCSのレベルで，マウスからのデータを受け取り，解析してマウス・カーソルの座標を移動させています（マウスからは移動量とボタンの状態しか送られてこない）．

マウスの処理は基本的にはタイマー割り込みルーチン内部で行なわれます．割り込みが周期的にかかったときに，割り込みルーチン内部でマウスにデータを要求し，送られてきたデータからマウス・カーソルの座標を移動させています．そのため，IOCSがマウス・データを受け取るために使っている割り込みを禁止すると，IOCSのマウス・データは更新されなくなります．

マウス・カーソルはテキストVRAMのプレーン2，3に表示されます．テキスト画面に表示されている文字によってマウス・カーソルのグラフィック

$70	マウス初期化
$71	マウスカーソル表示
$72	マウスカーソル消去
$73	マウスカーソルが表示されているか調べる
$74	マウスの移動量・ボタンの状態を調べる
$75	マウスカーソルの座標を調べる
$76	マウスカーソルの座標を指定する
$77	マウスカーソルの移動範囲を指定する
$78	マウスのボタンを離すまでの時間を調べる
$79	マウスのボタンを押すまでの時間を調べる
$7A	マウスカーソル・グラフィック・パターンを設定
$7B	マウスカーソル・グラフィック・パターンを選ぶ
$7C	マウスカーソル・グラフィック・パターンを複数選んでアニメーションにする
$7D	ソフトキーボード制御

[illegible]しないようにするため，テキスト・パレッ[illegible]から７までと，８から$Fまでにそれぞれ同じカラーコードを設定しています．

2.9 DMA

X68000は外部機器との大量データ転送にDMAを使っています．DMAは４つのチャンネルを持ち，それぞれが，メモリ・2HDフロッピーディスク間，メモリ・[illegible]間，メモリ・ADPCM間，メモリ・メモリ間のデータ転送を行ないます．

IOCS $8A～8Dがサポートしているのは，メモリ・メモリ間データ転送です．これらのIOCSによって，大量のデータのメモリ内移動が高速に行なえます．また，データ量が少ない場合は転送中にCPUが他の命令を実行できます（DMAがBUSを占有しないため）．

$8A	メモリ間 DMA 転送
$8B	メモリ間 DMA 転送 （チェーンテーブルによって複数のデータ・ブロックを転送）
$8C	メモリ間 DMA 転送 （アレイチェーンテーブルによって複数のデータ・ブロックを転送）
$8D	DMA の実行している IOCS を調べる

2.10 スプライト

X68000にはスプライト機能があり，ある程度まとまったキャラクタをドット単位で高速に移動させることができます．また，スプライト画面には，スプライトとは別にバックグラウンド画面があり，指定したPCGを並べて表示できます．表示色は，65536色中16色が使えます（スプライト画面全体で256色まで）．

なお，スプライト画面が表示できるのは表示画面のX方向が512，256ドットのときです．

$C0	スプライト画面の初期化
$C1	スプライト画面の表示
$C2	スプライト画面の非表示
$C3	PCG のクリア
$C4	PCG 定義
$C5	PCG 読み込み
$C6	スプライト・レジスタを定義して表示する
$C7	スプライト・レジスタ読み出し
$C8	BG スクロール・レジスタ定義
$C9	BG スクロール・レジスタ読み出し
$CA	BG コントロール・レジスタ設定
$CB	BG コントロール・レジスタ読み出し
$CC	BG テキストクリア
$CD	BG テキスト書き込み
$CE	BG テキスト読み出し
$CF	スプライトパレット定義

1．スプライト

スプライトは16×16ドット・パターンで，最大128個同時表示できます（X方向は32個まで）．表示位置や表示方法はスプライト・レジスタで指定します．パターンの一部が画面外に出るような表示もできます．パターンは指定したPCGデータ（16×16ドット）を使います．表示色はスプライトパレット・ブロックの指定したブロックにしたがいます．

2．バックグラウンド

バックグラウンド画面は画面モードによって決まる大きさのパターンが64×64個並んだもので，最大２面表示できます．表示位置はBGスクロール・レジスタで，表示方法はBGコントロール・レジスタで設定します．表示色は，65536色中16色が使えます（スプライト画面全体で256色まで）．

バックグラウンドはX方向のモードによって表示形体が違います．

〈256 ドット・モード時〉

バックグラウンド画面は２つ表示できます．１つのキャラクタは１つのPCGデータ（８×８ドット）を表示します．64×64キャラクタ分のBGテキストがあるのでバックグラウンド画面全体は512×512

ドットになります．そして，BGテキストの任意の位置からの256×256ドット分を表示できます．

<512 ドット・モード時>

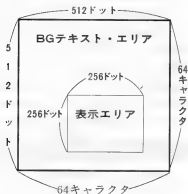
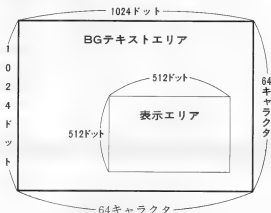
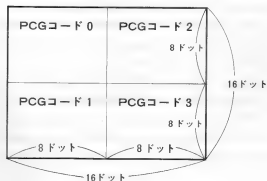
バックグラウンド画面は1つ表示できます．1つのキャラクタは1つのPCGデータ(16×16ドット)を表示します．64×64キャラクタ分のBGテキストがあるのでバックグラウンド画面全体は1024×1024ドットになります．そして，BGテキストの任意の位置からの512×512ドット分を表示できます．

3．PCG

スプライトやバックグラウンドのキャラクタのパターンはPCGに定義しておき，表示するときにPCGナンバーを指定することによって表示パターンを指定します．PCGはスプライト，バックグラウンド共通です．また，パターンの大きさは2種類ありますが，重複しています(4つの8×8ドット・パターンが1つの16×16ドット・パターンに相当する)．

4．スプライトパレット・ブロック

スプライトやバックグラウンドで指定する色はスプライトパレット・ブロックに定義しておきます．そして，表示するときにパレット・ブロックナンバーを指定することによって，パターンで指定される16種類のパレットコードにカラーコードをあてはめて表示します．パレット・ブロックは16個ありますが，0番のブロックはテキスト・パレットと同じものです．

ＢＧの表示位置

PCGコード0 (16×16)は4つのPCGコード0～3 (8×8ドット)と重複している．

2.11 文字フォント

X68000はROMにフォント・データを持ち，テキスト・グラフィックVRAMに転送・書き込みを行なって文字を表示しています．ROMに格納されているフォントの種類は次の６種類です．

8×８ドット　1/4角文字
8×16ドット　半角文字
16×16ドット　漢字・全角文字（第１・第２水準）
12×12ドット　1/4角文字
12×24ドット　半角文字
24×24ドット　漢字・全角文字（第１・第２水準）

IOCSではこれらのフォントに加えて，IOCSサブルーチン内で12×12ドット漢字・全角文字と６×12ドット半角文字を作成できます．

$0F	外字定義
$16	文字フォント・データ・アドレスを調べる
$19	文字フォントを読み出す
$38	外字フォント・データ・アドレス

IOCS $16，$19ではIOCSが作り出す大きさのフォントについてもROMにある大きさのフォントと同じように扱えます．よって，IOCSユーザーから見た場合，フォントは８種類あることになります．

2.12 その他のハードウェア

1．テレビコントロール機能

$0A	OPT.２によるテレビコントロール許可
$0B	OPT.２によるテレビコントロール禁止
$0C	テレビコントロール

X68000は専用ディスプレイ・テレビをソフトウェアでコントロールできます．また，主電源（背面のスイッチ）がONであれば，キーボードによってコントロールすることもできます．

2．RS232C

$30	RS232Cパラメーター設定
$31	受信データの数を読み出す
$32	受信
$33	受信データの有無を調べる
$34	送信可能であるか調べる
$35	送信

X68000には標準でRS232Cインターフェイスが１ポート装備されています(増設可)．このポートを使うことによって，各種周辺機器（例：モデム，他のコンピュータ）とデータのやりとりができます．

3．ジョイスティック

$3B	ジョイスティック状態を調べる

X68000には２つのアタリ社規格準拠ジョイスティック・ポートがあり，ゲームなどで使います．

4．プリンタ・インターフェイス

$3C	プリンタインターフェイス初期化
$3D	出力可能か調べる
$3E	直接出力
$3F	文字出力
$6F	プリンタによる割り込み制御

X68000には１つのセントロニクス社規格準拠のプリンタ・インターフェイスが装備されています．漢字プリンタを制御する場合，漢字モードの指定と解除を行なう必要がありますが，IOCS $3Fでは指定した文字コードを解析して，自動的に漢字モード指定・解除を行ないます．また，これらのIOCSは接続するプリンタによって処理を変更しないとうまく印字できない場合があります．そのため，Human68Kのプリンタ・ドライバなどは，処理アドレスを変更している場合があります．

プリンタ割り込みは，プリンタがBUZYでないときにかかります．

5．内部時計/アラーム機能

$50	日付バイナリ・データを日付 BCD データに変換
$51	内部時計を日付にセットにする（BCD データ）
$52	時刻バイナリ・データを時刻 BCD データに変換
$53	内部時計に時刻をセットする（BCD データ）
$54	内部時計から日付を読み込む
$55	日付 BCD データを日付バイナリ・データに変換
$56	内部時計から読み込む
$57	時刻 BCD データを時刻バイナリ・データに変換
$58	日付文字列を日付バイナリ・データに変換
$59	時刻文字列を時刻バイナリ・データに変換
$5A	日付バイナリ・データを日付文字列に変換
$5B	時刻バイナリ・データを時刻文字列に変換
$5C	曜日バイナリ・データを曜日文字列に変換
$5D	アラームの禁止，許可
$5E	アラームの時刻と処理内容（アドレス）を設定
$5F	アラームの時刻と処理内容（アドレス）を調べる

X68000にはバッテリーバックアップされた時計が装備されています．IOCS $50番台は，時計に対する処理をサポートしています．時計に対してはBCDデータしか使えないので，データ変換を行なうIOCSが各種あります．

アラームを使うことにより，指定した時間に専用ディスプレイテレビやX68000本体を制御できます．IOCS $5D～5Fによって設定ができます．

6．日本語処理用文字列処理

$A0	シフト JIS コードから JIS 漢字コードを計算
$A1	JIS 漢字コードからシフト JIS コードを計算
$A2	ANK コードから対応したシフト JIS コードを計算
$A3	ローマ字仮名変換
$A4	濁音処理
$A5	半濁点処理

X68000はIOCSのレベルでは漢字変換はできません．IOCSはローマ字変換のためのサブルーチンをサポートしています．

解説の例

$12 _HSVTORGB

IOCSコールナンバー　名前（XCのライブラリに対応しています）

引数

```
D1.l  HSVデータ
      bit 0～4       V (0～$1F) 明るさ
      bit 8～$S      S (0～$1F) 飽和度
      bit $10～$17  V (0～$BF) 色相
      以下のビットは0
```

引数と　その内容，フォーマット

返り値

```
D0.l  RGBカラーコード (%GGGGGRRR_RRBBBBBB0)
```

返値と　その内容，フォーマット

機能

HSVデータをRGBカラーコードに直します．

コール

```
moveq    #$12.D0
move.l   #$80120F,D1
trap     #15
```

実際の方法

サンプル・プログラム

P.444　IOCSサンプル6

$00　_B_KEYINP

引数

なし

返り値

D0.1　　スキャンコード×256＋内部コード

　　bit $F　　　　0のとき押された（リピートした），1のとき離された

　　bit 8～$E　　　キーの位置データ

　　bit 0～7　　　ASCIIコード，または0

61 62 | 63 64 65 66 67 | 68 69 6A 6B 6C | 5A 5B 5C | 5D 52 53 54
02 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F | 36 5E 37 | 3F 40 41 42
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D | 38 39 3A | 43 44 45 46
71 1E 1F 20 21 22 23 24 25 26 27 28 29 | 3B 3C 3D | 47 48 49 4A
70 2A 2B 2C 2D 2E 2F 30 31 32 33 34 70 | 3E | 4B 4C 4D 4E
5F 55 56 35 57 58 59 60 | 72 73 | 4F 50 51

キーコードは16進数

BREAK COPY | F1 F2 F3 F4 F5 | F6 F7 F8 F9 F10 | かな ローマ字 コード入力 | CAPS 記号入力 登録 HELP
ESC 1 2 3 4 5 6 7 8 9 0 - ^ ¥ BS | HOME INS DEL | CLR / * -
TAB Q W E R T Y U I O P @ [| ROLL UP ROLL DOWN UNDO | 7 8 9 +
CTRL A S D F G H J K L ; :] | ⇧ ⇦ ⇨ | 4 5 6 =
SHIFT Z X C V B N M , . / _ SHIFT | ⇩ | 1 2 3 ENTER
ひらがな XF1 XF2 XF3 XF4 XF5 全角 | OPT.1 OPT.2 | 0 , .

機能

キーデータを読み出します（電卓処理にも対応しています）．入力待ちをします．SHIFT，CTRL，OPT.1，OPT.2の4つのキーについては，放したときにもコードを返します．スキャンコードの値は対象となったキーの位置と，そのキーがどうなったかという情報（押された，リピートした，または離された）を示します．内部コードの値は入力されたASCII文字のコードで，SHIFT，CTRL，かな，CAPSキーによって修正されます．ASCII文字が定義されていないキーを押した場合は内部コードの値は0です．

キー入力はすべてバッファリングされています(ASCII文字が定義されていないキーのデータも含まれる)．特殊な働きを持っているキーは，その働きをした後に入力されます（COPY&OPT.1＋OPT.2など）．

コール

```
moveq   #0,D0
trap    #15
```

サンプル・プログラム

P.442　IOCSサンプル1

例

スペースキーが押された場合，D0.1は$3520になる

$01　_B_KEYSNS

引数

なし

返り値

D0.1　$10000＋スキャンコード×256＋内部コード

bit $10	キー入力がある場合は1
bit $F	1のとき押された(リピートした)，0のとき離された
bit 8～$E	キーの位置データ
bit 0～7	ASCIIコード，またはO

$0のときはキーは押されていない

61 62　63 64 65 66 67　68 69 6A 6B 6C　5A 5B 5C　5D 52 53 54
02 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F　36 5E 37　3F 40 41 42
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D　38 39 3A　43 44 45 46
71 1E 1F 20 21 22 23 24 25 26 27 28 29　3B 3C 3D　47 48 49 4A
70 2A 2B 2C 2D 2E 2F 30 31 32 33 34 70　3E　4B 4C 4D 4E
5F 55 56 35 57 58 59 60　72 73　4F 50 51

キーコードは16進数

BREAK COPY　F1 F2 F3 F4 F5　F6 F7 F8 F9 F10　かな ローマ字 コード入力　CAPS 記号入力 登録 HELP
ESC 1 2 3 4 5 6 7 8 9 0 - ^ ¥ BS　HOME INS DEL　CLR / * -
TAB Q W E R T Y U I O P @ [　ROLL UP　ROLL DOWN　UNDO　7 8 9 +
CTRL A S D F G H J K L ; :]　↵　⇦ ⇧ ⇨　4 5 6 =
SHIFT Z X C V B N M , . / _ SHIFT　⇩　1 2 3
ひらがな XF1 XF2　XF3 XF4 XF5 全角　OPT.1 OPT.2　0 , .　ENTER

機能

キーデータ・バッファ状態を調べます(電卓処理にも対応しています)．キーデータの内容はIOCS $00と同じです．

状態を調べるだけなので読み出しは行ないません．バッファの内容はそのまま残ります(IOCS $00で読み出せます)．

コール

```
        moveq   #1,D0
        trap    #15
```

例

CAPS OFF，かな OFF，SHIFT OFF，CTRL OFF の状態で「A」キーがバッファに入っている場合，D0.1は$11E61になります．

$02　_B_SFTSNS

引数

なし

返り値

D0.w　シフトキー状態（押している，またはLEDが点灯しているときに１になる）

bit 0	SHIFT	bit 8	かな
bit 1	CTRL	bit 9	ローマ字
bit 2	OPT.1	bit $A	コード入力
bit 3	OPT.2	bit $B	CAPS
bit 4	かな	bit $C	INS
bit 5	ローマ字	bit $D	ひらがな
bit 6	コード入力	bit $E	全角
bit 7	CAPS		

機能

シフトキー状態を調べます．返り値のビット$0から$7まではキー入力モード，ビット$8から$EまではLEDのモードを表わしています．

コール

```
	moveq	#2,D0
	trap	#15
```

例

「かな」ONで「SHIFT」が押されている場合，D0.wは$111になる

$03　_KEY_INIT

引数

D1.b　キーボードLED初期状態（１のときLEDが点灯する）

bit 0	かな
bit 1	ローマ字
bit 2	コード入力
bit 3	CAPS
bit 4	INS
bit 5	ひらがな
bit 6	全角
bit 7	0で固定

返り値

D0が破壊される

機能

キー入力関係を初期化します．キー・バッファがクリアされます．初期化後に，キーボードLEDとキー入力モードを指定した状態にします．通常は使用しません．

コール

```
	moveq	#3,D0
	move.b	#%100001,D1
	trap	#15
```

$04 _BITSNS

引数

D1.w　キーコード・グループナンバー（0～$E)

戻り値

D0.b　キーコード・グループ内のキー状態
(押しているとき，対応したビットが1になる)

機能

指定したキーが押されているかどうかを調べます．

コール

```
moveq    #4,D0
move.w   #5,D1
trap     #15
```

例

```
moveq    #4,D0
moveq    #2,D1
trap     #15
```

「TAB」キーが押されている場合，D0.bのbit 0が1となる

キーコード・グループナンバー	戻り値の bit							
	0	1	2	3	4	5	6	7
0		ESC	1	2	3	4	5	6
1	7	8	9	0	—	^	¥	BS
2	TAB	Q	W	E	R	T	Y	U
3	I	O	P	@	[	RET	A	S
4	D	F	G	H	J	K	L	;
5	:	]	Z	X	C	V	B	N
6	M	,	.	/	_	スペース	HOME	DEL
7	ROLL UP	ROLL DN	UNDO	←	↑	→	↓	CLR
8	(/)	(*)	(−)	(7)	(8)	(9)	(+)	(4)
9	(5)	(6)	(=)	(1)	(2)	(3)	ENTER	(0)
A	(,)	(.)	記号	登録	HELP	XF1	XF2	XF3
B	XF4	XF5	かな	ローマ字	コード入力	CAPS	INS	ひらがな
C	全角	BREAK	COPY	F1	F2	F3	F4	F5
D	F6	F7	F8	F9	F10			
E	SHIFT	CTRL	OPT. 1	OPT. 2				

注（　）はテンキー

$05　_SKEYSET

引数

D1.1　スキャンコード

bit 7　1のとき押された（リピートした），0のとき離された

bit 0〜6　キーの位置データ

そのほかのビットは0

返り値

D0が破壊される

機能

ソフト的にキー入力を発生させます．キーデータはサブCPUが送るデータと同じ形式です．

コール

```
moveq   #5,D0
move.b  #$11,D1
trap    #15
```

サンプル・プログラム

P.442　IOCSサンプル2

$06　_LED_ON

引数

D1.b　キーボードLED状態（1のときLEDが点灯する）

bit 0　かな

bit 1　ローマ字

bit 2　コード入力

bit 3　CAPS

bit 4　INS

bit 5　ひらがな

bit 6　全角

返り値

D0が破壊される

機能

キーボードLEDのモードを設定します．キー入力モードは変更されません．

コール

```
moveq   #6,D0
moveq   #%1111010,D1
trap    #15
```

例

```
moveq   #6,D0
moveq   #$11,D1
trap    #15
```

この場合，「かな」と「INS」のLEDが点灯，その他のLEDは消灯する

$07 _LEDMOD

引数

なし

返り値

D0が破壊される

機能

キーボードLEDのモードをキー入力モードに合わせます.

コール

```
        moveq   #7,D0
        trap    #15
```

$08 _KEY_REPI

引数

D1.b　　キー・リピート開始間隔（0～$F)

返り値

D0が破壊される

機能

キー・リピート開始までの時間を設定します. 100×D1.w+200msになります.

コール

```
        moveq   #8,D0
        move.w  #2,D1
        trap    #15
```

サンプル・プログラム

P.443　IOCSサンプル3

$09 _KEY_REP2

引数

D1.b　　キー・リピート間隔（0～$F）

返り値

D0が破壊される

機能

キーリピート間隔の時間を設定します．5×D1.w²＋30msになります．

コール

```
moveq   #9,D0
move.w  #6,D1
trap    #15
```

サンプル・プログラム

P.443　IOCSサンプル3

$0A _OPT2_TV_CTRL

引数

なし

返り値

D0が破壊される

機能

OPT.2キーのテレビコントロールができるようにします．

コール

```
moveq   #$A,D0
trap    #15
```

$0B

引数

なし

返り値

D0が破壊される

機能

OPT.2キーのテレビコントロールができないようにします．シフトキーによるコントロールはできます．

コール

```
moveq   #$B,D0
trap    #15
```

$0C　_TVCTRL

引数

D1.1　テレビコントロール・コマンド

1	ボリュームを上げる
2	ボリュームを下げる
3	ボリュームを普通にする
4	チャンネルコール
5	テレビ画面（初期化&リセット）
6	音声ミュート
7	電源ON
8	テレビ・コンピュータ画面切り替え
9	テレビ・外部入力の切り替え，またはコンピュータ表示モードの切り替え
$A	コントラストを普通にする
$B	チャンネルアップ
$C	チャンネルダウン
$D	電源OFF
$E	電源ON・OFF切り替え
$F	スーパー・インポーズ&コント・ラストダウン・解除の切り替え
$10	チャンネル1
$11	チャンネル2
$12	チャンネル3
$13	チャンネル4
$14	チャンネル5
$15	チャンネル6
$16	チャンネル7
$17	チャンネル8
$18	チャンネル9
$19	チャンネル10
$1A	チャンネル11
$1B	チャンネル12
$1C	テレビ画面（初期化&リセット）
$1D	コンピュータ画面
$1E	スーパーインポーズ&コントラスト・ダウン
$1F	スーパーインポーズ&コントラスト・ノーマル
+$20	電源ONの後，上記の機能を実行する

返り値

D0が破壊される

機能

専用CRTをコントロールします．

コール

```
moveq   #$C,D0
move.w  #$D,D1
trap    #15
```

サンプル・プログラム

P.443　IOCSサンプル4

$0D　_LEDMOD

引数

D1.l　モードをセットするキー&LED

D1.l		モードをセットするキー&LED
	0	かな
	1	ローマ字
	2	コード入力
	3	CAPS
	4	INS
	5	ひらがな
	6	全角
D2.b	0	OFF
	1	ON（点灯）

返り値

D0が破壊される

機能

指定したキー入力モードと，キーボードLEDのモードを設定します．

コール

```
moveq   #$D,D0
moveq   #0,D1
moveq   #1,D2
trap    #15
```

例

```
moveq   #$D,D0
moveq   #3,D1
moveq   #1,D2
trap    #15
```

この場合，「CAPS」キーのLEDが点灯し，CAPS ONになります．

$0E　_TGUSEMD

引数

D1.b　0　グラフィック画面
　　　1　テキスト画面
D2.b　0　使っていない
　　　1　システムで使っている（ソフトキーボード・電卓）
　　　2　アプリケーションで使っている
　　　3　アプリケーションで使った後で壊れたままである
　　　−1　どのモードかを調べる

返り値

D0.l　D1で指定した画面のモード（0～3）
　　　−1のときエラー

機能

テキスト・グラフィック画面の使用状態を設定します．

テキスト画面をアプリケーションで使う指定をすると，マウス・カーソル＆ソフトキーボード＆電卓が表示されません．これらのものがあらかじめ表示されていた場合は，マウス・カーソルはそのまま表示され，ソフトキーボード＆電卓は消えます．グラフィック画面やテキスト画面を使う場合は，このIOCSコールによって，その画面を使うことを宣言するようにします．

このIOCSコールをすべてのアプリケーションが使うことにより，アプリケーション間での画面の重複使用が避けられます．

コール

```
moveq   #$E,D0
move.b  #1,D1
move.b  #2,D2
trap    #15
```

$0F　　_DEFCHR

引数

D1.1　上位16ビット　文字の大きさ

　　0,8　16*16, 8*16

　　$C　24*24, 12*24

　下位16ビット　シフトJISコード，JIS漢字コード

　　全角:$EB9F～$EBFC ($7621～$767E)

　　　$EC40～$EC7E, $EC80～$EC9F ($7721～$777E)

　　半角:$F400～$F5FF

A1.1　外字パターン・データ・アドレス

■12×12

■6×12

bit 7　　　　bit 2	bit 1　　bit 0
＋0	使用しない
＋1	使用しない
⋮	⋮
＋$B	使用しない

■16×16

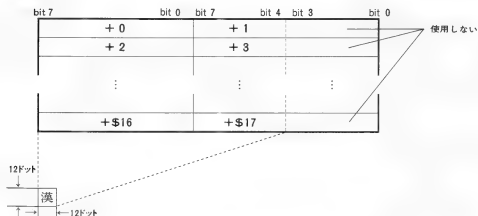
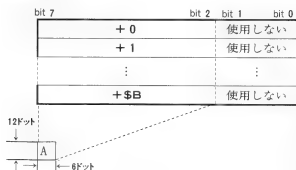
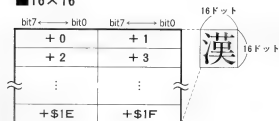
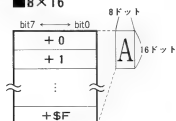
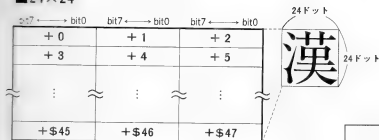
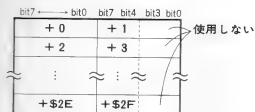
bit7←→bit0	bit7←→bit0
＋0	＋1
＋2	＋3
⋮	⋮
＋$1E	＋$1F

16ドット　漢　16ドット

■8×16

■24×24

■12×24

返り値

D0.l　　−1のときエラー

機能

外字を設定します.

コール

```
        moveq   #$F,D0
        move.l  #$8*65536+$7621
        lea     gaiji__data,A1
        trap    #15
```

$10　_CRTMOD

引数

D1.w　　画面モード・ナンバー

D1.W の値	表示エリア	グラフィック画面の構成			同期周波数	スプライト
0	512×512	1024×1024	16色	1枚	31kHz	可
1	512×512	1024×1024	16色	1枚	15kHz	可
2	256×256	1024×1024	16色	1枚	31kHz	可
3	256×256	1024×1024	16色	1枚	15kHz	可
4	512×512	512×512	16色	4枚	31kHz	可
5	512×512	512×512	16色	4枚	15kHz	可
6	256×256	512×512	16色	4枚	31kHz	可
7	256×256	512×512	16色	4枚	15kHz	可
8	512×512	512×512	256色	2枚	31kHz	可
9	512×512	512×512	256色	2枚	15kHz	可
$A	256×256	512×512	256色	2枚	31kHz	可
$B	256×256	512×512	256色	2枚	15kHz	可
$C	512×512	512×512	65536色	1枚	31kHz	可
$D	512×512	512×512	65536色	1枚	15kHz	可
$E	256×256	512×512	65536色	1枚	31kHz	可
$F	256×256	512×512	65536色	1枚	15kHz	可
$10	768×512	1024×1024	16色	1枚	31kHz	不可
$11	1024×424	1024×1024	16色	1枚	24kHz	不可
$12	1024×848	1024×1024	16色	1枚	24kHz	不可

返り値

D0.l　　現在のモード・ナンバー（D1.w=－1）
　　　　それ以外のときは破壊される

機能

画面モードを設定します．テキストVRAMの$E00000～$E3FFFFをクリアし，テキスト・パレットを初期化します．グラフィック画面とスプライト画面はクリアされずに無表示モードになります．

D1.w=ナンバー＋$100のときは画面モードの切り替えだけを行ないます．D1.w=－1のときは現在のモードナンバーを返します．

コール

```
moveq    #$10,D0
move.w   #$14,D1
trap     #15
```

サンプル・プログラム

P.444　IOCSサンプル5

$11　_CONTRAST

引数

D1.b　0～$F　コントラストの設定
　　　－1　現在のコントラストを調べる
　　　－2　システム設定値に戻す

返り値

D0.l　変更前（D1.b＝0～$F，－2），または現在（D1.b＝－1）の値

機能

コントラストを変更します．

コール

```
moveq    #$11,D0
move.b   #$4,D1
trap     #15
```

$12　_HSVTORGB

引数

D1.l　HSVデータ
　　　bit 0～4　　V（0～$1F）明るさ
　　　bit 8～$C　 S（0～$1F）飽和度
　　　bit $10～$17　H（0～$BF）色相
　　　他のビットは0

返り値

D0.l　RGBカラーコード（%GGGGGRRR__RRBBBBB0）
　　　－1のときエラー

機能

HSVデータをRGBカラーコードに直します．返り値は輝度が0となっています．

コール

```
moveq    #$12,D0
move.l   #$80120F,D1
trap     #15
```

サンプル・プログラム

P.444　IOCSサンプル6

$13　_TPALET

引数

D1.b　テキスト・パレットコード

0	テキスト・カラー（0）
1	テキスト・カラー（1）
2	テキスト・カラー（2）
3	テキスト・カラー（3）
4～7	ソフト・キーボード&電卓カラー（0） テキスト・カラー（4～8）すべて
8～$F	ソフト・キーボード&電卓カラー（1） テキスト・カラー（9～$F）すべて

D2.l　0～$FFFF　カラーコード（0～$FFFF）

0～$FFFF	カラーコード（0～$FFFF）
−1	現在のカラーコードを調べる
−2	システム設定値に戻す

返り値

D0.l　D2.l＝−1のとき現在のカラーコード

機能

テキスト・パレットを設定します．D1.b＝4～$Fのときはそれぞれに対応したテキスト・カラーがすべて設定されます．これは，IOCSレベルでサポートしているソフトウェアキーボードや電卓，マウス・カーソルを背景の影響をまったく受けないように表示させるためです．

コール

```
moveq   #$13,D0
move.b  #2,D1
move.l  #$1234,D2
trap    #15
```

サンプル・プログラム

IOCSサンプル6

$14　_TPALET2

引数

D1.b　テキスト・パレットコード

　0～$F　テキスト・カラー（0～$F）

D2.l　0～$FFFF　カラーコード（0～$FFFF）

　−1

現在のカラーコードを調べる

返り値

D0.l　D2.l＝−1のとき現在のカラーコード

　−1のときエラー

機能

テキスト・パレットを設定します．IOCS $13とは違って，テキスト・カラーごとに独立

させて設定できます.

コール

```
moveq   #$14,D0
move.b  #3,D1
move.l  #$1234,D2
trap    #15
```

サンプル・プログラム

P.445　IOCSサンプル7

$15　_TCOLOR

引数

D1.b　アクセスされるテキストVRAMプレーン

0	テキスト・プレーン1	($E00000～$E1FFFF)
1	テキスト・プレーン1	($E00000～$E1FFFF)
2	テキスト・プレーン2	($E20000～$E3FFFF)
4	テキスト・プレーン3	($E40000～$E5FFFF)
8	テキスト・プレーン4	($E60000～$E7FFFF)

返り値

D0が破壊される

機能

IOCSコール($1A～$1C)でアクセスされるテキストVRAMプレーンを指定します.アクセス終了後はテキスト・プレーン1を指定してください.

コール

```
moveq   #$15,D0
move.b  #4,D1
trap    #15
```

サンプル・プログラム

P.446　IOCSサンプル8

$16　_FNTADR

引数

D1.w　　シフトJISコード，JIS漢字コード

D2.l　　文字のサイズ

6	12*12, 6*12 Dot
8	16*16 8*16 Dot
$C	24*24 12*24 Dot

返り値

D0.l　　バッファアドレス（スーパーバイザ・エリア）

D1.w　　(文字パターンのX方向のバイト数)－1

D2.w　　(文字パターンのY方向のドット数)－1

■12×12

■6×12

bit 7 … bit 2	bit 1 … bit 0
＋0	使用しない
＋1	使用しない
⋮	⋮
＋$B	使用しない

12ドット　A　6ドット

■16×16

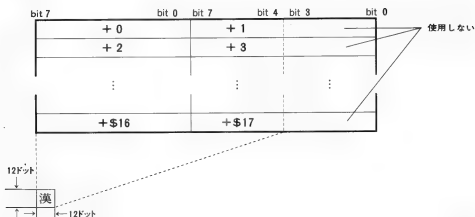
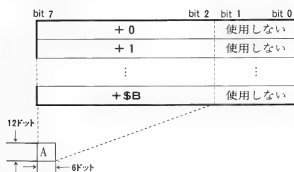
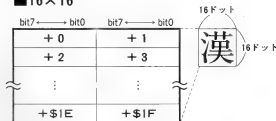
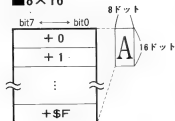
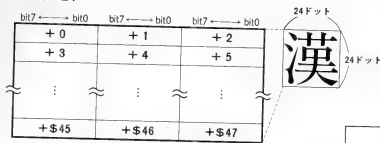
bit7 ←→ bit0	bit7 ←→ bit0
＋0	＋1
＋2	＋3
⋮	⋮
＋$1E	＋$1F

16ドット　漢　16ドット

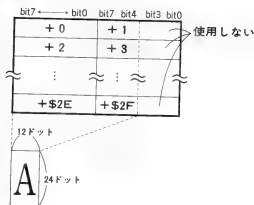
■8×16

■24×24

bit7 ←→ bit0	bit7 ←→ bit0	bit7 ←→ bit0
+0	+1	+2
+3	+4	+5
⋮	⋮	⋮
+$45	+$46	+$47

24ドット
漢
24ドット

1バイト

■12×24

機能

漢字フォント・パターンのアドレスを調べます．通常ROMのアドレスになりますが，外字や12×12ドット，または6×12ドットのフォントのアドレスはRAMのアドレスになることがあります．よってこの場合はフォントのデータがいつまでも残るとは限りません．

コール

```
moveq   #$16,D0
move.w  #$2121,D1
moveq   #8,D2
trap    #15
```

サンプル・プログラム

P.456　IOCSサンプル8

例

```
moveq   #$16,D0
move.w  #'電',D1
moveq   #6,D2
trap    #15
```

この場合，D1.w=1，D2.w=$B，D0.l=フォント・データのアドレス　がセットされる．

$17　_VRAMGET

引数

D1.w　(バッファのX方向のバイト数)－1

D2.w　(バッファのY方向のドット数)－1

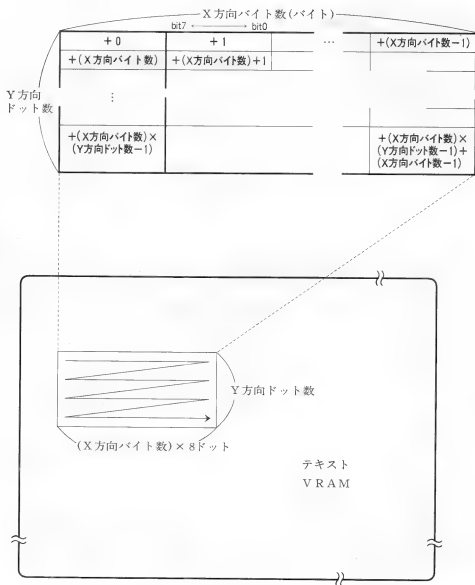
D3.l　(テキストVRAMのX方向のバイト数)－(D1.w)－1

A1.l　バッファの先頭アドレス

A2.l　テキストVRAMの先頭アドレス

返り値

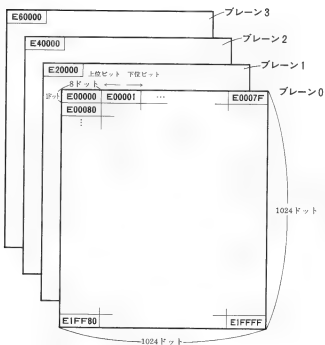
D0, D1, D2, A1, A2が破壊される

機能

指定した範囲のテキストVRAMのデータをバイト単位で読み込みます。

コール

```
moveq    #$17,D0
move.w   #$F,D1
move.w   #$F,D2
move.l   #$6F,D3
lea      bufadd,A1
movea.l  #$E01000,A2
trap     #15
```

例

```
moveq    #$17,D0
move.w   #1,D1
move.w   #$F,D2
move.l   #$80-1-1,D3
lea      bufadd,A1
movea.l  #$E00000,A2
trap     #15
```

$18　　VRAMPUT

引数

D1.w　　(データのX方向のバイト数)－1

D2.w　　(データのY方向のドット数)－1

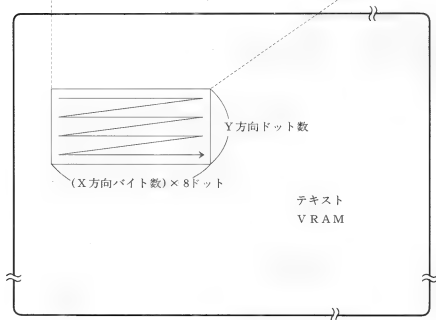
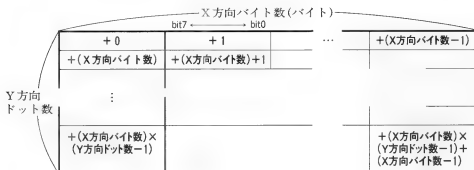
D3.l　　(テキストVRAMのX方向のバイト数)－ (D1.w)－1

A1.l　　データの先頭アドレス

A2.l　　テキストVRAMの先頭アドレス

返り値

D0, D1, D2, A1, A2が破壊される

機能

指定した範囲のテキストVRAMにデータをバイト単位で書き込みます.

コール

```
        moveq    #$17,D0
        move.w   #$F,D1
        move.w   #$F,D2
        move.l   #$6F,D3
        lea      bufadd,A1
        movea.l  #$E01000,A2
        trap     #15
```

サンプル・プログラム

P.446 IOCSサンプル8

SRAMの初期化方法

SRAMにはswitch.xで設定するメモリ・スイッチの内容をはじめとして，システムが正常に働くための値が納められています．このSRAMの内容が何かの拍子で書き変わってしまうと，いろいろと問題が起きてきます．正しいはずのプログラムで妙なエラーが頻発したり，目的のデバイスから正しくブートできなくなったりした場合は，SRAMの内容がおかしくなってしまった可能性があります．

このような場合は，次のようなプログラムを入力して，実行してみてください．短いプログラムですからデバッガから直接入力・実行してもよいでしょう．

```
        move.b   #$31, $E8E00D
        clr.b    $ED0000
        clr.b    $E8E00D
        dc.w     $FF00
```

実行後，いちどリセットを行なうと，SRAMの内容が初期化されます．

メモリ・スイッチの内容などもすべて出荷時の状態に戻ってしまいますから，再度設定し直す必要があります．

$19 _FNTGET

引数

D1.l　上位16ビット　文字の大きさ

6	12*12, 6*12
8	16*16, 8*16
$C	24*24, 12*24

　　下位16ビット　シフトJISコード，JIS漢字コード

A1.l　外字パターン読み込みバッファアドレス

返り値

(A1).w　文字パターンのX方向のドット数

2(A1).w　文字パターンのY方向のドット数

4(A1).b 以降にフォント・データがセットされている

■12×12

■6×12

bit 7 ～ bit 2	bit 1 ～ bit 0
+0	使用しない
+1	使用しない
⋮	⋮
+$B	使用しない

■16×16

bit7 ←→ bit0	bit7 ←→ bit0
+0	+1
+2	+3
⋮	⋮
+$1E	+$1F

16ドット　漢　16ドット

■8×16

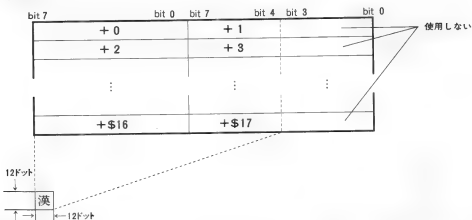
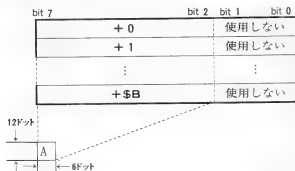
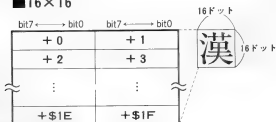
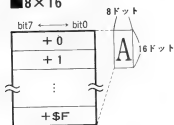
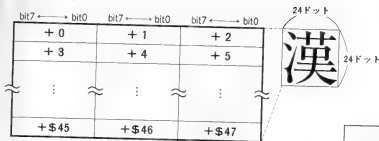
■24×24

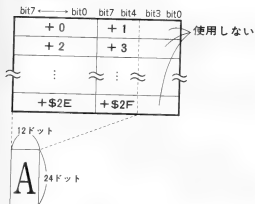
bit7 ←→ bit0	bit7 ←→ bit0	bit7 ←→ bit0
＋0	＋1	＋2
＋3	＋4	＋5
⋮	⋮	⋮
＋$45	＋$46	＋$47

24ドット

漢

24ドット

⇩

1バイト

■12×24

機能

指定した文字のフォント・パターンを読み出します.

コール

```
moveq    #$19,D0
move.l   #$C*65536+$2121,D1
lea      bufadd,A1
trap     #15
```

サンプル・プログラム

P.447　IOCSサンプル9

$1A　_TEXTGET

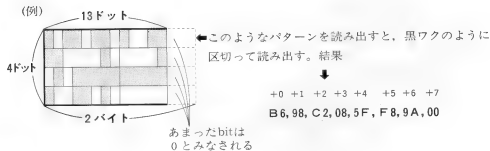
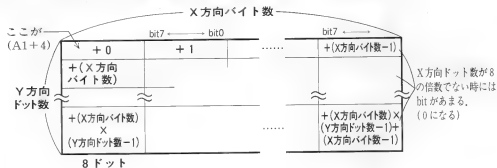
引数

D1.w　　X ドット座標
D2.w　　Y ドット座標
A1.l　　データ・バッファの先頭アドレス
(A1).w　　パターンのX方向のドット数
2(A1).w　　パターンのY方向のドット数
4(A1).b　　以降が読み込みバッファ

返り値

D0が破壊される

テキストVRAMドット単位読み出しデーターフォーマット

機能

指定した範囲のテキストVRAMの内容をドット単位で読み出します.

コール

```
moveq   #$1A,D0
move.w  #$57,D1
move.w  #$45,D2
lea     bufadd,A1
trap    #15
```

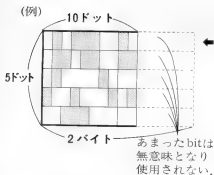
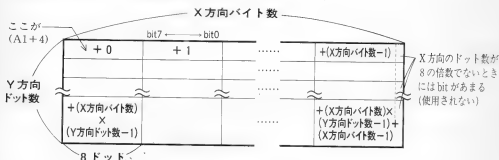
サンプル・プログラム

P.447　IOCSサンプル10

$1B _TEXTPUT

引数

D1.w	Xドット座標
D2.w	Yドット座標
A1.l	パターン・データの先頭アドレス
(A1).w	パターンのX方向のドット数
2(A1).w	パターンのY方向のドット数
4(A1).b	以降が書き込みデータ

テキストVRAMドット単位書き込みデーターフォーマット

←このようなパターンを書き込む場合，パターンデーターは黒ワクのように区切って作る．結果

↓

+0	+1	+2	+3	+4
5C	80	6C	C0	42
+5	**+6**	**+7**	**+8**	**+9**
00	91	40	38	00

返り値

D0が破壊される

機能

指定した範囲のテキストVRAMにドット・パターンを書き込みます.

コール

```
moveq   #$1B,D0
move.w  #$57,D1
move.w  #$45,D2
lea     bufadd,A1
trap    #15
```

サンプル・プログラム

P.447　IOCSサンプル10

$1C　_CLIPPUT

引数

D1.w　　Xドット座標
D2.w　　Yドット座標
A1.l　　パターン・データの先頭アドレス
A2.l　　クリッピング・データの先頭アドレス
(A1).w　　パターンのX方向のドット数
2(A1).w　　パターンのY方向のドット数
4(A1).b　　以降が書き込みデータ
(A2).w　　表示領域の左端のX座標
2(A2).w　　表示領域の上端のY座標
4(A2).w　　表示領域の右端のX座標
6(A2).w　　表示領域の下端のY座標

テキストVRAMドット単位書き込みデーターフォーマット

返り値

D0が破壊される

機能

指定した範囲のテキストVRAMにドット・パターンを書き込みます．クリッピングを行ないます．

コール

```
moveq   #$1B,D0
move.w  #$57,D1
move.w  #$45,D2
lea     bufadd,A1
trap    #15
```

サンプル・プログラム

P.447　IOCSサンプル9

$1D　_SCROLL

引数

D1.w　0　グラフィック画面(0)を設定
　　　1　グラフィック画面(1)を設定
　　　2　グラフィック画面(2)を設定
　　　3　グラフィック画面(3)を設定
　　　4　グラフィック画面(0)を調べる
　　　5　グラフィック画面(1)を調べる
　　　6　グラフィック画面(2)を調べる
　　　7　グラフィック画面(3)を調べる
　　　8　テキスト画面を設定
　　　9　テキスト画面を調べる

D2.w　Xドット座標

D3.w　Yドット座標

返り値

D0.l　上位16ビット　Xドット座標（設定前（D1.w＝0～3,8）
　　　または現在の（D1.w＝4～7,9））
　　　下位16ビット　Yドット座標（設定前（D1.w＝0～3,8）
　　　または現在の（D1.w＝4～7,9））

機能

テキスト&グラフィック画面の表示座標を設定します．グラフィック画面についてはすべての位置に設定できます．テキスト画面については，Y座標はすべての位置に，X座標は表示画面が実画面に収まる位置に設定できます．

コール

```
moveq   #$1D,D0
move.w  #$3,D1
move.w  #$100,D2
move.w  #$100,D3
trap    #15
```

サンプル・プログラム

P.449　IOCSサンプル11

$1E　_B_CURON

引数

なし

返り値

D0が破壊される

機能

カーソルの一時停止を解除します．

コール

```
moveq   #$1E,D0
trap    #15
```

$1F　_B_CUROFF

引数

なし

返り値

D0が破壊される

機能

カーソルを一時停止します．

コール

```
moveq   #$1F,D0
trap    #15
```

$20　_B_PUTC

引数

D1.w　　シフトJISコード
　　　　(全角文字の場合は第1バイトと第2バイトを別べつに送ってもよい)

返り値

D0.l　　上位16ビット　　カーソルの移動後の桁位置
　　　　下位16ビット　　カーソルの移動後の行位置

機能

カーソル位置に文字を表示します．テキスト画面のプレーン0＆1の表示範囲内に書き込まれます．右端へくると次の行に移ります（最下行の場合はスクロールします）．

右端に全角文字表示させせようとすると，まず半角スペースが表示されて次の行に移ってから指定した全角文字を表示します．

文字の属性はIOCS $22で指定します．

コール

```
moveq   #$20,D0
move.w  #'A',D1
trap    #15
```

$21　_B_PRINT

引数

A1.l　文字列データ先頭アドレス（データの最後は０）

返り値

D0.l　上位16ビット　カーソルの移動後の桁位置
　　　下位16ビット　カーソルの移動後の行位置

A1.l　文字例データの最後の0のアドレス

機能

カーソル位置に文字を表示します．テキスト画面のプレーン０＆１の表示範囲内に書き込まれます．右端へくると次の行に移ります（最下行の場合はスクロールします）．

右端に全角文字を表示させようとすると，まず半角スペースが表示されて次の行に移ってから指定した全角文字を表示します．

文字の属性はIOCS $22で指定します．

コール

```
moveq   #$21,D0
lea     string,A1
trap    #15
```

$22　_B_COLOR

引数

D1.w　カラーコード

0	テキスト・パレット（０）	通常黒
1	テキスト・パレット（１）	通常水色
2	テキスト・パレット（２）	通常黄色
3	テキスト・パレット（３）	通常白
4＋上記	それぞれの強調色	
8＋上記	それぞれのリバース	
$C＋上記	それぞれの強調色のリバース	
−1	現在のカラーを調べる	

返り値

D0.l　変更前のカラー（D1.w＝0-$F）
　　　現在のカラー（D1.w＝−1）

機能

表示文字のカラー属性を指定します．IOCS $20，$21で表示される文字に使用されます．

コール

```
moveq   #$22,D0
move.w  #$1,D1
trap    #15
```

$23　_B_LOCATE

引数

D1.w　桁位置

　　−1　現在の座標を調べる

D2.w　行位置

返り値

D0.l　上位16ビット　カーソルの移動前（D1.w＝−1のときは現在）の桁位置

　　　下位16ビット　カーソルの移動前（D1.w＝−1のときは現在）の行位置

機能

カーソルを移動します．

コール

```
moveq   #$23,D0
move.w  #$3,D1
move.w  #$10,D2
trap    #15
```

$24　_B_DOWN_S

引数

なし

返り値

D0が破壊される

機能

カーソルを１行下へ移動します．いちばん下の行の場合はスクロール・アップします．

コール

```
moveq   #$24,D0
trap    #15
```

$25　_B_UP_S

引数

なし

返り値

D0が破壊される

機能

カーソルを１行上へ移動します．１番上の行の場合はスクロール・ダウンします．

コール

```
moveq   #$25,D0
trap    #15
```

$26　_B_UP

引数

D1.b　　行数（0は1とみなされる）

返り値

D0が破壊される

機能

カーソルを指定した行数だけ上へ移動します(スクロールはしない)．指定した行数移動できない場合は移動しません．

コール

```
        moveq   #$26,D0
        move.b  #$3,D1
        trap    #15
```

$27　_B_DOWN

引数

D1.b　　行数（0は1とみなされる）

返り値

D0が破壊される

機能

カーソルを指定した行数だけ下へ移動します(スクロールはしない)．指定した行数移動できない場合は表示範囲の最下行まで移動します．

コール

```
        moveq   #$27,D0
        move.b  #$2,D1
        trap    #15
```

$28　_B_RIGHT

引数

D1.b　　桁数（0は1とみなされる）

返り値

D0が破壊される

機能

カーソルを指定した桁数だけ右へ移動します(スクロールはしない)．指定した桁数移動できない場合は表示範囲の右端へ移動します．

コール

```
        moveq   #$28,D0
        move.b  #$8,D1
        trap    #15
```

$29　_B_LEFT

引数

D1.b　　桁数（0は1とみなされる）

返り値

D0が破壊される

機能

カーソルを指定した桁数だけ左へ移動します(スクロールはしない)．指定した行数移動できない場合は表示範囲の左端へ移動します．

コール

```
moveq   #$29,D0
move.b  #$4,D1
trap    #15
```

$2A　_B_CLR_ST

引数

D1.b　　消去エリアの指定

　0　　カーソル位置から最終行右端まで
　1　　先頭行左端からカーソル位置まで
　2　　画面全体（カーソルは先頭行左端へ移動）

返り値

D0が破壊される

機能

画面の複数の行を消去します．スクロールはしません．

コール

```
moveq   #$2A,D0
move.b  #$1,D1
trap    #15
```

$2B _B_ERA_ST

引数

D1 消去エリアの指定

0 カーソル位置から行の右端まで

1 行の左端からカーソル位置まで

2 カーソルのある行すべて

返り値

D0が破壊される

機能

行の複数の桁を消去します．スクロールはしません．

コール

```
moveq   #$2B,D0
move.b  #$2,D1
trap    #15
```

$2C _B_INS

引数

D1.b 挿入行数（0は1とみなされる）

返り値

なし

機能

カーソル位置に複数行挿入します．カーソルは左端に移動します．カーソル位置より下の行はスクロールします（カーソルのある行も含む）．

コール

```
moveq   #$2C,D0
move.b  #$5,D1
trap    #15
```

$2D _B_DEL

引数

D1.b 削除行数（0は1とみなされる）

返り値

なし

機能

カーソル位置から複数行削除します．カーソル位置より下の行はスクロールします．

コール

```
moveq   #$2D,D0
move.b  #$2,D1
trap    #15
```

$2E _B_CONSOL

引数

D1.l　上位16ビット　X方向の表示開始ドット位置（16の倍数で1008以下）
　　　下位16ビット　Y方向の表示開始ドット位置（4の倍数で1020以下）
　　　－1　変化させない
D2.l　上位16ビット　(X方向の表示桁数)－1（0～127）
　　　下位16ビット　(Y方向の表示桁数)－1（0～63）
　　　－1　変化させない

返り値

D0 が破壊される
D1.l　上位16ビット　変更前のX方向の表示開始ドット位置
　　　下位16ビット　変更前のY方向の表示開始ドット位置
D2.l　上位16ビット　(変更前のX方向の表示桁数)－1
　　　下位16ビット　(変更前のY方向の表示桁数)－1

機能

テキスト画面の表示範囲を設定します．カーソルは先頭行左端に移動します．IOCS $20～2Dで扱う座標は設定された表示範囲の左上が（0,0）となり，表示範囲内に作用します．ただし，スクロール・画面クリアは表示範囲外に影響が出ます．

コール

```
moveq   #$2E,D0
move.l  #$100*65536+$10,D1
move.l  #$30*65536+$20,D2
trap    #15
```

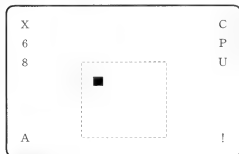
サンプル・プログラム

P.449　IOCSサンプル12

点線内がIOCS $2Eで指定した表示範囲

■はカーソル

上のようなときに，カーソル位置に文字を表示すると**下図a**のようになる．また画面クリアすると**下図b**のようになる．

(a)スクロール後

(b)画面クリア

$2F _B_PUTMES

引数

D1.b　カラーコード

0	テキスト・パレット（0）	通常黒
1	テキスト・パレット（1）	通常水色
2	テキスト・パレット（2）	通常黄色
3	テキスト・パレット（3）	通常白
4＋上記	それぞれの強調色	
8＋上記	それぞれのリバース	
$C＋上記	それぞれの強調色のリバース	

D2.w　絶対桁位置

D3.w　絶対行位置

D4.w　(表示桁数)－1

A1.l　文字列データ先頭アドレス（データの最後は0）

返り値

D0，D4が破壊される

D2.w　次の桁位置

A1.l　文字列データの最後の0のアドレス

機能

文字列を表示します．表示桁数以上は表示されません（全角文字が半分だけ表示できる場合は，その文字は表示されません．半角分スペースが空くことになります）．行が変わる場合はそれ以降の文字は表示されません．スクロールもしません．

コール

```
moveq   #$2F,D0
move.b  #$3,D1
move.w  #$10,D2
move.w  #$E,D3
move.w  #$8,D4
lea     string,A1
trap    #15
```

$30 _SET232C

RS232Cの設定値

−1　　現在のモードを調べる

RS232C設定フォーマット

```
bit   F E D C B A 9 8 7 6 5 4 3 2 1 0
D1.w  <-><-><-><->0 0 0 0 0 0<->
```

ボーレート

000	75
001	150
010	300
011	600
100	1200
101	2400
110	4800
111	9600

XON/XOFF処理

0	処理しない
1	処理する

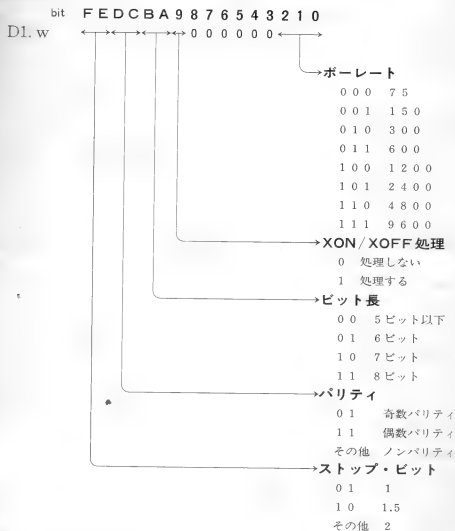
ビット長

00	5ビット以下
01	6ビット
10	7ビット
11	8ビット

パリティ

01	奇数パリティ
11	偶数パリティ
その他	ノンパリティ

ストップ・ビット

01	1
10	1.5
その他	2

返り値

D0.l　変更前のモード (D1.w <>−1)

　　　現在のモード (D1.w=−1)

機能

RS232Cのパラメータを設定します．

コール

```
        moveq   #$30,D0
        move.w  #%00001111_00000100,D1
        trap    #15
```

$31 _LOF232C

引数

なし

返り値

D0.w　　バッファ内のデータ数

機能

RS232Cの受信バッファ内のデータ数を調べます．

コール

```
moveq   #$31,D0
trap    #15
```

$32 _INP232C

引数

なし

返り値

D0.b　　受信データ

機能

RS232Cの受信データを読み出します．受信データがない場合は受信するまで待ちます．

コール

```
moveq   #$32,D0
trap    #15
```

$33 • _ISNS232C

引数

なし

返り値

D0.l　　$10000＋受信データ
　　　　0 のときは受信データはない

機能

RS232Cにデータが送られているかを調べます．読み出しはIOCS $32で行ないます．

コール

```
moveq   #$33,D0
trap    #15
```

$34 _OSNS232C

引数

[illegible]

返り値

[illegible]　0のときは送信できない
　　　4のときは送信できる

機能

RS232Cに送信ができるかどうかを調べます．バッファが空で，XONでないときに送信可能になります．

コール

```
        moveq   #$34,D0
        trap    #15
```

$35 _OUT232C

引数

D1.b　送信データ

返り値

D0が破壊される

機能

RS232Cにデータを送信します．

コール

```
        moveq   #$35,D0
        move.b  #'A',D1
        trap    #15
```

$36 _MS_JMPADD

引数

D1.l　処理アドレス
D2.w　カウンター初期値（1～$FFFF，0）

返り値

D0が破壊される

機能

マウスからデータを受信したときにジャンプするアドレスを設定します．通常はマウスIOCSで使うデータの書き換えを行なうアドレスが設定されています．このアドレスにジャンプしてくるときはA1.lが示すアドレスからの3バイトにマウスからの受信データがセットされています．

カウンターの値を大きくすると，ジャンプする間隔が長くなります（1が最短）．

コール

```
        moveq   #$36,D0
        move.l  #jobadd,D1
        move.w  #1,D2
        trap    #15
```

$37 _ESC_JMPADD

引数

D1.1　処理アドレス
　　　0にすると拡張解除（RTSのアドレスがセットされる）

返り値

D0が破壊される

機能

エスケープシーケンスのうち，「ESC[>」系統の拡張処理アドレスを設定します．このアドレスにジャンプしてくる場合，D0.wに「ESC[>」の次の2バイト・データ，A0.lに「ESC[>」の「ESC[>」のアドレスがセットされています．処理ルーチンの最後は「RTS」にします．

コール

```
moveq   #$37,D0
move.l  #jobadd,D1
trap    #15
```

$38 _FONT_ADD

引数

D1.l　フォント・アドレス
D2.l　外字グループナンバー（0～5）

	サイズ	JISコード
0	16×16ドット	($2C21～$2D7E)
1	16×16ドット	($7621～$777E)
2	8×16ドット	($F400～$F5FF)
3	24×24ドット	($2C21～$2D7E)
4	24×24ドット	($7621～$777E)
5	12×24ドット	($F400～$F5FF)

返り値

D0.l　変更前のアドレス
　　　−1のときエラー

機能

外字フォント・データ・アドレスを変更します．

コール

```
moveq   #$38,D0
move.l  #gaijiadd,D1
moveq   #2,D2
trap    #15
```

$39 _BEEP_PCM

引数

D1.L　BEEP音PCMデータ先頭アドレス

D2.W　データのバイト数

返り値

D0.L　以前のBEEP音PCMデータ先頭アドレス

D1.W　以前のデータのバイト数

D2　破壊される

機能

BEEP音用PCMデータの設定を行ないます．

コール

```
moveq    #$39,D0
move.l   #pcmadd,D1
move.w   #pcmlen,D2
trap     #15
```

TIMER.X の未公開オプション

Human68K Ver2.0xのバックグラウンド機能のサンプルとして用意されているTIMER.Xには未公開のオプションが用意されています．

●/X TIME <file_name>

指定した時刻に，<file_name>で指定したプログラムを起動します．

●/KILL

Timer.xの常駐を解除し，メモリを開放します．

$3A _PRN_PAR

引数

A1.1　　プリンタIOCS用パラメータデータ・アドレス
　　　　0のときROMの初期値になる

プリンタ・エスケープシーケンス設定

オフセット	設　　定
+0.L	VRAMプレーン0アドレス($E00000)
+4.W	Y方向ドット数÷12−1(41)
+6.L	$FFFF00
+10.W	Y方向ドット数÷24−1(21)
+12.L	$FF0000
+16.W	X方向ドット数÷8−1(96)
+18.B〜	漢字モード指定コード
+26.B〜	漢字モード解除コード
+34.B〜	LFのコード
+38.B〜	改行幅　12/180インチ
+44.B〜	改行幅　1/180インチ
+50.B〜	改行幅　8/180インチ
+56.B〜	改行幅　4/180インチ
+62.B〜	ビットイメージ出力コード(1536ドット出力)
+70.B〜	ビットイメージ出力コード(768ドット出力)
+78.B〜	ビットイメージ出力コード(18ドット出力)
+86.B〜	ビットイメージ出力コード(36ドット出力)
+94.B〜	0…MSB〜LSB，1…LSB〜MSBの順で出力
+95.B	?
+96.B	?
+97.B	モード

(+18.B〜+94.B〜)
+0.bコードバイト数
+1.bコード
+n.b0

返り値

D0が破壊される

機能

　プリンタIOCSで使うパラメータを定義します．

コール

```
        moveq   #$3A,D0
        lea     prndat,A1
        trap    #15
```

$3B _JOYGET

引数

D1.w　読み込むジョイスティック番号

　0　ジョイスティック 1

　1　ジョイスティック 2

返り値

D0.b　ジョイスティックの状態（ビットが0のとき入力あり）

　bit 0　上

　bit 1　下

　bit 2　左

　bit 3　右

　bit 5　トリガー2

　bit 6　トリガー1

機能

ジョイスティックの状態を調べます．

コール

```
moveq   #$3B,D0
move.w  #$1,D1
trap    #15
```

例

```
moveq   #$3B,D0
move.w  #0,D1
trap    #15
```

ジョイスティック1が上になっている場合，D0.bのbit 0が0になる．

$3C _INIT_PRN

引数

D1.w　上位8ビット　(1ページの行数)−1

　　　　　　　　　−1 ページ指定なし

　　　下位8ビット　(1行の文字数)−1

　　　　　　　　　−1 行指定なし

返り値

D0.l　0のとき出力できない

　　　$20のとき出力できる

機能

プリンタ関係の初期化をします．D1レジスタによる指定はIOCS $3F用です．初期化と同時に，出力可能かどうか調べます．

コール

```
moveq   #$3C,D0
move.w  #$3FF,D1
trap    #15
```

$3D　_SNSPRN

引数

なし

返り値

D0.l　0のとき出力できない
　　$20のとき出力できる

機能

プリンタに出力可能かどうか調べます.

コール

```
moveq   #$3D,D0
trap    #15
```

$3E　_OUTLPN

引数

D1.b　出力データ

返り値

D0が破壊される

機能

プリンタにデータを出力します. データはそのままプリンタに送られます. 通常はプリンタに出力が完了するまで戻りません.

コール

```
moveq   #$3E,D0
move.w  #$20,D1
trap    #15
```

$3F　_OUTPRN

引数

D1.b　出力文字（シフトJISコード）

返り値

D0が破壊される

機能

プリンタに文字を出力します. 2バイト文字の場合は2回に分けてコールしてください（上位, 下位の順）. 文字のコードは変換された後プリンタに出力されます. 通常はプリンタに出力が完了するまで戻りません.

コール

```
moveq   #$3F,D0
move.w  #'A',D1
trap    #15
```

$40　_B_SEEK

上位８ビット　PDA
下位８ビット　MODE
目的シーク位置

MODE 指定フォーマット

■2HD フロッピーディスク

PDA	
$90	2HD フロッピーディスク・ドライブ 0（本体内蔵）
$91	1（本体内蔵）
$92	2（外部）
$93	3（外部）

MODE	意　　味
bit 6	0＝FM（単密）モード　1＝MFM（倍密）モード
bit 5	0＝リトライしない　1＝10回リトライする（最初の５回はそのまま，後の５回は，リキャリブレートとシークをやりなおしてからリトライする）
bit 4	0＝シークせずに実行　1＝シーク後に実行
bit 3	} 0 に固定
bit 0	

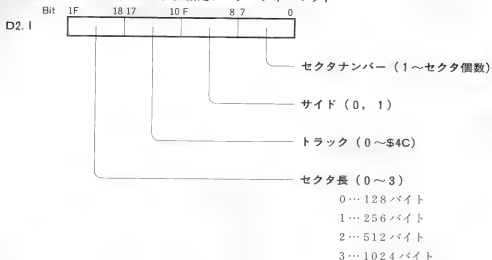
■ハードディスク

PDA	
$80	ハードディスクドライブ 0
$81	1
⋮	⋮
$8F	15

MODE は意味を持たない.

・2HDフロッピーディスク　位置指定データーフォーマット

```
     Bit  1F       18 17      10 F      8 7       0
D2.l     |                                          |
          |         |          |          |
          |         |          |          └── セクタナンバー（1～セクタ個数）
          |         |          └──────────── サイド（0，1）
          |         └─────────────────────── トラック（0～$4C）
          └───────────────────────────────── セクタ長（0～3）
```

セクタ長（0～3）
0…128バイト
1…256バイト
2…512バイト
3…1024バイト

・ハードディスク

D2.l　256バイト単位のレコード番号

返り値

PDAが$90～$93のとき

D0.l　bit $10～$17　コマンド終了後のトラックナンバー
　　　bit $18～$1F　ステータス・レジスタ0

PDAが$80～$8Fのとき

D0.l　ステータス
　　　正の数のとき正常
　　　負の数のときは下位8ビットがSCSIからのエラーコード

機能

ディスクの指定位置までシークします．

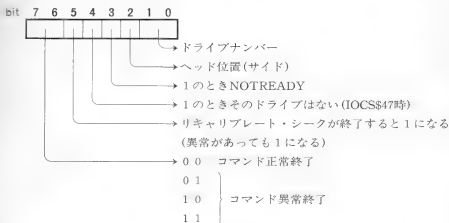
コール

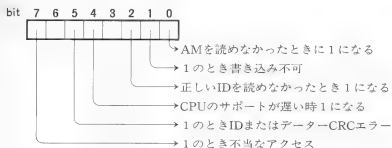
```
moveq   #$40,D0
move.w  #$9070,D1
move.l  #$034C0105,D2
trap    #15
```

例

```
moveq   #$40,D0
move.w  #$9170,D1
move.l  #$03000001,D2
trap    #15
```

この場合，2HDドライブ1のトラック0，サーフェス0のセクタ1にシークする．

ステータスレジスタ2HDフロッピーディスク

ST 0（ステータス・レジスタ 0）

ST 1（ステータス・レジスタ 1）

ST 2（ステータス・レジスタ 2）

$41 _B_VERIFY

引数

D1.w	上位８ビット	PDA
	下位８ビット	MODE
D2.l	目的チェック位置	
D3.l	チェックするバイト数	
A1.l	チェックするデータの先頭アドレス	

返り値

PDA が $90～$93のとき

D0.l	bit 0～7	コマンド終了後のトラックナンバー
	bit 8～$F	ステータス・レジスタ2
	bit $10～$17	ステータス・レジスタ1
	bit $18～$1F	ステータス・レジスタ0
D2.l	最後に読み込んだセクタの位置	
D3.l	０になる	
A1.l	比較データの次のアドレス	

PDAが$80～$8Fのとき

D0.l　ステータス
　正の数のとき正常
　負の数のときは下位８ビットがSCSIからのエラーコード

機能

データの比較チェックをします．2HDの場合はデータが一致した場合はステータス・レジスタ２のbit3が１になります．一致しない場合はステータス・レジスタ２のbit2が１になります．

コール

```
moveq   #$41,D0
move.w  #$9070,D1
move.l  #$03000105,D2
move.l  #$400,D3
lea     data_add,A1
trap    #15
```

サンプル・プログラム

P.449　IOCSサンプル13

$42　_B_READDI

引数

[illegible]	上位８ビット	PDA
	下位８ビット	MODE
[illegible]	目的読み出し位置	
[illegible]	読み出しバイト数	
[illegible]	読み出したデータのバッファアドレス	

返り値

[illegible]	bit 0～7	コマンド終了後のトラックナンバー
	bit 8～$F	ステータス・レジスタ2
	bit $10～$17	ステータス・レジスタ1
	bit $18～$1F	ステータス・レジスタ0
[illegible]	最後に読み込んだセクタの位置	
[illegible]	0になる	
[illegible]	読み出したデータの次のアドレス	

機能

[illegible]HDの診断のためのディスク読みだしをします．通常の読み出しの場合は使いません．

コール

```
moveq    #$42,D0
move.w   #$9070,D1
move.l   #$03150107,D2
move.l   #$800,D3
lea      data_buf,A1
trap     #15
```

”＼”はどこにいった？

知っている方も多いと思いますが，ファンクションコール・レベルでは，パスの区切り記号として”￥”（もしくは”／”）の他に”＼”を使うことができます．

ところが，いくつか存在するHumanのバージョンの中には，これが使えないものも存在するのです．それはバージョン2.00．

もともとマニュアルに書かれていない機能ですし，このような事情も発生してしまったわけですから，パスの区切り記号は”￥”以外使わないようにするのが無難だと思います．

$43　_B_DSKINI

引数

D1.w　上位８ビット　PDA
　　　下位８ビット　MODE

PDAが$80～$8Fのとき

A1.l　0　デフォルトになる
　　　0 以外　アサイン・ドライブパラメータのデータ・アドレス

PDAが$90～$93のとき

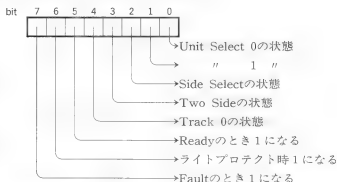
A1.l　0　デフォルトになる
　　　0 以外　SPECIFYコマンドのデータ・アドレス

D2.w　0　デフォルトになる
　　　0 以外　モーターオフまでの時間 (1＝0.01Sec)

返り値

PDAが$90～$93のとき

D0.l　bit $18～$1F　ステータス・レジスタ3

ST3（ステータスレジスタ）

PDAが$80～$8Fのとき

D0.l　ステータス
　　　正の数のとき正常
　　　負の数のときは下位８ビットがSCSIからのエラー・コード

機能

ディスク関係の初期化を行ないます.

コール

```
moveq   #$43,D0
move.w  #3000,D2
lea     data_add,A1
trap    #15
```

$44　_B_DRVSNS

引数

D1.w　　上位8ビット　　PDA
　　　　下位8ビット　　$00

返り値

PDAが$90～$93のとき

D0.l　　bit $18～$1F　　ステータス・レジスタ3

PDAが$80～$8Fのとき

D0.l　　ステータス
　　　　正の数のとき正常
　　　　負の数のときは下位8ビットがSCSIからのエラー・コード

機能

ディスク・ステータスの調査を行ないます.

コール

```
moveq   #$44,D0
move.w  #$8000,D1
trap    #15
```

X68000ユーザーはパソ通しなさい！

いかにX68000の登場以来月日がたったとはいえ，その間，月に1度しか発行されない月刊雑誌たちのそのまた一部でしかないX68000関係の情報を集めたところで，X68000の秘めた底知れない魅力を知りつくすことは不可能です.

都市部に住んでいて，X68000に好意的なショップで情報収集ができるユーザーはまだいいでしょうが，地方に住んでいるユーザーは情報的に孤立した状態にあるのではないでしょうか.

そんな方に，私は強力にお勧めします，いや，命令します．パソコン通信をお始めなさい.

新鮮な情報，良質なPDS/フリーウェア，そしてなによりも大勢の「仲間」が得られます.

ことに，**PEKIN**，**梁山泊**などのようなX68000中心に異様な盛り上がりを見せるネットは要チェックです．どちらも大規模とは言えないネットですが，ことX68000に関することであれば，どこにも負けないパワーが凝縮されています．ぜひいちどはアクセスさせてもらいましょう．ただし，前述のとおり大規模なネットではないので，回線数が限られています．他のユーザーのことも考えて，時間を有効に使ってアクセスするようにしてください.

モデムを接続したあなたのX68Kは，それまでとは違う顔をみせてくれることでしょう.

$45　_B_WRITE

引数

D1.w　上位8ビット　PDA
　　　下位8ビット　MODE
D2.l　目的書き込み位置
D3.l　書き込みバイト数
A1.l　書き込みデータ先頭アドレス

返り値

PDAが$90～$93のとき
D0.l　bit 0～7　コマンド終了後のトラックナンバー
　　　bit 8～$F　ステータス・レジスタ2
　　　bit $10～$17　ステータス・レジスタ1
　　　bit $18～$1F　ステータス・レジスタ0
D2.l　最後に書き込んだセクタの位置
D3.l　0になる
A1.l　書き込みデータの次のアドレス

PDAが$80～$8Fのとき
D0.l　ステータス
　　　正の数のとき正常
　　　負の数のときは下位8ビットがSCSIからのエラー・コード

機能

ディスクにデータを書き込みます.

コール

```
        moveq   #$45,D0
        move.w  #$9070,D1
        move.l  #$03010101,D2
        move.l  #$C00,D3
        lea     data_add,A1
        trap    #15
```

例

```
        moveq   #$45,D0
        move.w  #$9070,D1
        move.l  #$03000001,D2
        move.l  #$C00,D3
        lea     data_add,A1
        trap    #15
```

この場合，data_addからの$C00バイトが2HDディスクに書き込まれる．返り値は，D0.l＝ステータス，D2.l＝$03000003，D3.l＝0，A1.l＝data_add＋$C00になる．

$46　_B_READ

引数

D1.w　上位8ビット　PDA
　　　下位8ビット　MODE
D2.l　目的読み出し位置
D3.l　読み出しバイト数
A1.l　読み出したデータのバッファ・アドレス

返り値

PDAが$90～$93のとき

D0.l　bit 0～7　コマンド終了後のトラック・ナンバー
　　　bit 8～$F　ステータス・レジスタ2
　　　bit $10～$17　ステータス・レジスタ1
　　　bit $18～$1F　ステータス・レジスタ0
D2.l　最後に読み込んだセクタの位置
D3.l　0になる
A1.l　読み出したデータの次のアドレス

PDAが$80～$8Fのとき

D0.l　ステータス
　　　正の数のとき正常
　　　負の数のときは下位8ビットがSCSIからのエラー・コード

機能

ディスクからデータを読み出します.

コール

```
        moveq   #$46,D0
        move.w  #$9070,D1
        move.l  #$03050006,D2
        move.l  #$1000,D3
        lea     data_add,A1
        trap    #15
```

サンプル・プログラム

P.449　IOCSサンプル13

$47　_B_RECALI

引数

D1.w　上位８ビット　PDA
　　　下位８ビット　$00

2HDの場合にのみ，次の設定が可能

D1.w　上位８ビット　$90〜$93 (PDA)
　　　下位８ビット　$FF

返り値

PDAが$90〜$93のとき (D1.b=０)

D0.l　bit $10〜$17　コマンド終了後のトラック・ナンバー
　　　bit $18〜$1F　ステータス・レジスタ０

PDAが$90〜$93のとき (D1.b=−1)

D0.l　bit $10〜$17　コマンド終了後のトラック・ナンバー
　　　bit $18〜$1F　ステータス・レジスタ０
　　　　(bit 4が１の場合はそのドライブはない)

PDAが$80〜$8F のとき (D1.b=０)

D0.l　ステータス
　　　正の数のとき正常
　　　負の数のときは下位８ビットがSCSIからのエラー・コード

機能

D1.b=０ならトラック０へシークします(2HD&ハードディスク). D1.b=−1のときは強制レディ状態での調査を行ないます. この場合はドライブの有無が調査できます (2HDのみ).

コール

```
	moveq	#$47,D0
	move.w	#$9000,D1
	trap	#15
```

$48　_B_ASSIGN

引数

D1.w　上位８ビット　$80〜$8F(PDA)

D2.l　レコード番号

D3.l　インタリーブ・コード

A1.l　代替トラック指定データ・アドレス
　　　(A1).b　代替トラック番号上位バイト
　　　1(A1).b 代替トラック番号中位バイト
　　　2(A1).b 代替トラック番号下位バイト
　　　3(A1).b 0

返り値

D0.l　　ステータス
　　　　正の数のとき正常
　　　　負の数のときは下位8ビットがSCSIからのエラー・コード

機能

ハードディスクの代替トラックを設定します.

$49　_B_WRITED

引数

D1.w　　上位8ビット　　PDA
　　　　下位8ビット　　MODE
D2.l　　目的書き込み位置
D3.l　　書き込みバイト数
A1.l　　書き込みデータ先頭アドレス

返り値

D0.l　　bit 0～7　　　コマンド終了後のトラック・ナンバー
　　　　bit 8～$F　　　ステータス・レジスタ2
　　　　bit $10～$17　ステータス・レジスタ1
　　　　bit $18～$1F　ステータス・レジスタ0
D2.l　　最後に書き込んだセクタの位置
D3.l　　0になる
A1.l　　書き込みデータの次のアドレス

機能

2HDディスクへ破損データを書き込みます. 通常の書き込みでは使いません.

コール

```
moveq   #$49,D0
move.w  #$9070,D1
move.l  #$03100004,D2
move.l  #$400,D3
lea     data_add,A1
trap    #15
```

$4A　_B_READID

引数

D1.w　上位８ビット　PDA
　　　下位８ビット　MODE
D2.l　目的読み出し位置（トラックとサーフェスの指定のみ）

返り値

D0.l　bit 0～7　コマンド終了後のトラック・ナンバー
　　　bit 8～$F　ステータス・レジスタ2
　　　bit $10～$17　ステータス・レジスタ1
　　　bit $18～$1F　ステータス・レジスタ0

2HDのID情報を読み出します．

コール

```
        moveq   #$4A,D0
        move.w  #$9070,D1
        move.l  #$110200,D2
        trap    #15
```

$4B　_B_BADFMT

引数

D1.w　$80～$8F (PDA)
D2.l　レコード番号
D3.l　インタリーブ・コード

返り値

D0.l　ステータス
　　　正の数のとき正常
　　　負の数のときは下位８ビットがSCSIからのエラー・コード

機能

ハードディスクの破壊トラックを使用不能にします．

$4C　_B_READDL

引数

D1.w	上位８ビット	PDA
	下位８ビット	MODE
D2.l	目的読み出し位置	
D3.l	読み出しバイト数	
A1.l	読み出したデータのバッファ・アドレス	

返り値

D0.l	bit 0～7	コマンド終了後のトラック・ナンバー
	bit 8～$F	ステータス・レジスタ２
	bit $10～$17	ステータス・レジスタ１
	bit $18～$1F	ステータス・レジスタ０
D2.l	最後に読み込んだセクタの位置	
D3.l	０になる	
A1.l	読み出したデータの次のアドレス	

機能

2HDディスクから破損データを読み出します．通常の読み出しでは使いません．

コール

```
moveq   #$4C,D0
move.w  #$9070,D1
move.l  #$03150106,D2
move.l  #$1000,D3
lea     data_add,A1
trap    #15
```

$4D　_B_FORMAT

引数

D1.w　上位８ビット　PDA
　　　下位８ビット　MODE
D2.l　目的フォーマット位置
D3.l　インターリーブコード（PDA＝$80～$8F）
D3.l　IDデータ・バイト数（PDA＝$90～$93）
A1.l　IDデータ先頭アドレス（PDA＝$90～$93）

D3.l バイト

A１	トラックナンバー（０～$４C）	１番目のセクタ
A１＋１	サイド（０，１）	
A１＋２	セクタ番号（１～セクタの個数）	
A１＋３	セクタ長（０－３） ０…１２８バイト １…２５６バイト ２…５１２バイト ３…１０２４バイト	
A１＋４ A１＋７		２番目のセクタ
A１＋n A１＋n＋３		最後のセクタ

※セクタの個数（MFM）

２５６バイト	２６コ	が普通
５１２バイト	１５コ	
１０２４バイト	８コ	

返り値

PDAが$90～$93のとき

D0.l　bit 0～7　ステータス・レジスタ０
　　　bit 8～$F　ステータス・レジスタ１
　　　bit $10～$17　ステータス・レジスタ２
　　　bit $18～$1F　コマンド終了後のトラック・ナンバー

PDAが$80～$8Fのとき

D0.l　ステータス
　　　正の数のとき正常
　　　負の数のときは下位８ビットがSCSIからのエラー・コード

機能

ディスクをフォーマットします．

ニール

```
        moveq   #$4D,D0
        move.w  #$9070,D1
        move.l  #$03000000,D2
        move.l  #4*8,D3
        lea     id_data,A1
        trap    #15
```

サンプル・プログラム

P.450　IOCSサンプル14

WP.X の未公開機能

本体購入時に付属してきたWP.Xにはいくつかの未公開機能が存在します．

●起動時に編集ファイル指定

WP.XをCOMMAND.Xから起動する場合，コマンドライン上から編集するファイルを指定して起動できます．

A>WP filename.SWP

拡張子".SWP"は必ず必要です．".swp"ではダメで，大文字で指定するようにしてください．

●ファンクション・キーに短文登録

文中の登録したい文字列をマウスでドラックして範囲指定状態にします．この状態で［SHIFT］を押しながら［F1］～［F10］を押すと，そのキーに指定した文字列が登録されます．以降，［F1］～［F10］を押すことで，ワンキーで文字列を入力できます．

ただし，登録された短文はディスクなどに記録されることはありませんので，WP.Xを起動するたびに登録を行なう必要があります．

● CUT の Undo / Redo

CUTした直後，［UNDO］キーを押すとCUTした部分をUNDOします．もういちど押すと再びCUT直後の状態に戻ります．

ちなみに，初期ユーザーの間でウケた，『X68000の紹介』は，WP.X Ver1.01で削除されてしまったようです(ごぞんじない方は，最下行の"X68000"という文字をクリックしてみてください)．

$4E　_B_DRVCHK

引数

D1.w　上位８ビット　PDA
　　　下位８ビット　$00

D2.w　機能番号

0	現在の状態を調べる
1	イジェクトする（禁止されているとイジェクトできない）
2	イジェクト禁止１
3	イジェクト許可１
4	ディスクがセットされていないときLED点滅
5	ディスクがセットされていないときLED消灯
6	イジェクト禁止２（OSで使用）
7	イジェクト許可２（OSで使用）
8	前回のチェック以来イジェクトしたかどうかを調べる（OSで使用）

返り値

D0.b　状態（D2.w＝0-7）

bit 0	誤挿入
bit 1	挿入
bit 2	Not Ready
bit 3	Protect
bit 4	ユーザー禁止
bit 5	バッファあり
bit 6	イジェクト禁止
bit 7	LED点滅

D0.b　状態（D2.w=8）

1	イジェクトしていない
−1	イジェクトされている

機能

2HDドライブの状態を設定，調査します．

コール

```
moveq	#$4E,D0
move.w	#$9000,D1
move.w	#$4,D2
trap	#15
```

サンプル・プログラム

P.451　IOCSサンプル15

$4F　_B_EJECT

引数

D1.w　上位8ビット　PDA
　　　下位8ビット　$00

返り値

PDAが$90～$93のとき

D0が破壊される

PDAが$80～$8Fのとき

D0.l　ステータス
　　　正の数なら正常
　　　負の数のときは下位8ビットがSCSIからのエラーコード

機能

イジェクト（2HD）またはシッピング（ハードディスク）します．イジェクト禁止状態でも実行します．

コール

```
moveq   #$4F,D0
move.w  #$9000,D1
trap    #15
```

$50　_DATEBCD

引数

D1.l　日付データ（バイナリ）
　　　$0y_yy_mm_dd
　　　　$yyy　年バイナリ（1980～2079）
　　　　$mm　月バイナリ（1～12）
　　　　$dd　日バイナリ（1～31）

返り値

D0.l　日付データ（BCD）
　　　$Ww_YY_MM_DD
　　　　$W　閏年カウンタ（0～3）
　　　　$w　曜日カウンタ（0～6:0＝日曜，6＝土曜）
　　　　$YY　年BCD（0～$99:0＝1980年，$99＝2079年）
　　　　$MM　月BCD（1～$12）
　　　　$DD　日BCD（1～$31）
　　－1　日付がおかしい場合（存在しない日付）

機能

日付データを内部時計にセットできる形式に変換します．

コール

```
moveq   #$50,D0
move.l  #$07C50310,D1
trap    #15
```

例

```
moveq   #$50,D0
move.l  #$07C50308,D1
trap    #15
```

この場合，D0.l＝$13090308となる．

$51　_DATESET

引数

D1.l　日付データ (BCD)

　　$Ww_YY_MM_DD

$W	閏年カウンタ（0～3）
$w	曜日カウンタ（0～6:0＝日曜，6＝土曜）
$YY	年BCD（0～$99:0＝1980年，$99＝2079年）
$MM	月BCD（1～$12）
$DD	日BCD（1～$31）

返り値

D0が破壊される

機能

日付データを内部時計にセットします．

コール

```
moveq   #$50,D0
move.l  #$23_09_03_08,D1
trap    #15
```

$52　_TIMEBCD

引数

D1.l　時刻データ（バイナリ）

　　$00_hh_mm_ss

$hh	時バイナリ（0～23）
$mm	分バイナリ（0～59）
$ss	秒バイナリ（0～59）

返り値

D0.l　時刻データ (BCD)

　　$01_HH_MM_SS

$HH	時BCD（0～$23）
$MM	分BCD（0～$59）
$SS	秒BCD（0～$59）

機能

時刻データを内部時計にセットできる形式に変換します．

コール

```
moveq   #$52,D0
move.l  #$00021530,D1
trap    #15
```

例

```
moveq   #$52,D0
move.l  #$000C2238,D1
trap    #15
```

この場合，D0.l＝$01123456 となる．

$53 _TIMESET

引数

D1.l　時刻データ (BCD)

$01_HH_MM_SS

$HH	時BCD (0～$23)
$MM	分BCD (0～$59)
$SS	秒BCD (0～$59)

返り値

D0が破壊される

機能

時刻データを内部時計にセットします.

コール

```
moveq   #$53,D0
move.l  #$01_12_34_56,D1
trap    #15
```

$54 _DATEGET

引数

なし

返り値

D0.l　日付データ (BCD)

$0w_YY_MM_DD

$w	曜日カウンタ (0～6:0＝日曜,…6＝土曜)
$YY	年BCD (0～$99:0＝1980年,…$99＝2079年)
$MM	月BCD (1～$12)
$DD	日BCD (1～$31)

機能

内部時計から日付を読み出します.

コール

```
moveq   #$54,D0
trap    #15
```

サンプル・プログラム

P.451　IOCSサンプル16

$55　_DATEBIN

引数

D1.l　日付データ (BCD)

$0w_YY_MM_DD

$w	曜日カウンタ（0～6:0＝日曜，6＝土曜）
$YY	年BCD（0～$99:0＝1980年，$99＝2079年）
$MM	月BCD（1～$12）
$DD	日BCD（1～$31）

返り値

D0.l　日付データ（バイナリ）

$wy_yy_mm_dd

$w	曜日カウンタ（0～6:0＝日曜，6＝土曜）
$yyy	年バイナリ（1980～2079）
$mm	月バイナリ（1～12）
$dd	日バイナリ（1～31）

機能

BCD表現の日付データをバイナリ表現に直します.

コール

```
moveq   #$55,D0
move.l  #$03_09_03_08,D1
trap    #15
```

例

```
moveq   #$55,D0
move.l  #$03090308,D1
trap    #15
```

この場合，D0.l＝$37C50308となる.

サンプル・プログラム

P.451　IOCSサンプル16

$56　_TIMEGET

引数

なし

返り値

D0.l　時刻データ (BCD)

　$00_HH_MM_SS

　　$HH　時BCD (0～$23)

　　$MM　分BCD (0～$59)

　　$SS　秒BCD (0～$59)

機能

内部時計から時刻を読み出します.

コール

```
moveq   #$56,D0
trap    #15
```

サンプル・プログラム

P.451　IOCSサンプル16

$57　_TIMEBIN

引数

D1.l　時刻データ (BCD)

　$00_HH_MM_SS

　　$HH　時BCD (0～$23)

　　$MM　分BCD (0～$59)

　　$SS　秒BCD (0～$59)

返り値

D0.l　時刻データ (バイナリ)

　$00_hh_mm_ss

　　$hh　時バイナリ (0～23)

　　$mm　分バイナリ (0～59)

　　$ss　秒バイナリ (0～59)

機能

BCD表現の時刻データをバイナリ表現に直します.

コール

```
moveq   #$57,D0
move.l  #$00_12_34_56,D1
trap    #15
```

例

```
moveq   #$57,D0
move.l  #$00123456,D1
trap    #15
```

この場合, D0.l＝$000C2238となる.

サンプル・プログラム

P.452　IOCSサンプル17

$58 _DATECNV

引数

A1.1　日付データ・アドレス（文字列：最後は0）

文字列のフォーマットは（1989年3月8日の場合）

'1989/03/08',0または'89/03/08',0

区切りは'－'でもよい。

返り値

D0.1　日付データ（バイナリ）

$0y_yy_mm_dd

$yyy	年バイナリ	(1980～2079)
$mm	月バイナリ	(1～12)
$dd	日バイナリ	(1～31)

機能

日付を表わす文字列の示している日付を読み出します。

コール

```
        moveq   #$58,D0
        lea     date_add,A1
        trap    #15
```

例

```
        moveq   #$58,D0
        lea     datadd,A1
        trap    #15

datadd  dc.b    '89/03/08',0
```

この場合，D0.1=$07C50308 となる。

$59 _TIMECNV

引数

A1.l　時刻データ・アドレス（文字列：最後は0）

文字列のフォーマットは（12時34分56秒の場合）

'12:34:56',0

返り値

D0.l　時刻データ（バイナリ）

$00_hh_mm_ss

$hh	時バイナリ	(0～23)
$mm	分バイナリ	(0～59)
$ss	秒バイナリ	(0～59)

機能

時刻を表わす文字列の示している時刻を読み出します.

コール

```
moveq   #$59,D0
lea     time_add,A1
trap    #15
```

例

```
        moveq   #$59,D0
        lea     datadd,A1
        trap    #15

datadd  dc.b    '12:34:56',0
```

この場合，D0.l＝$000C2238となる.

$5A _DATEASC

引数

D1.l　　日付データ（バイナリ）

　　$Fy_yy_mm_dd

　　　　$yyy　　年バイナリ（1980～2079）

　　　　$mm　　月バイナリ（1～12）

　　　　$dd　　日バイナリ（1～31）

　　　　Fは変換したい文字列フォーマットを指定する

　　　　　F＝0　　'1989/09/02',0

　　　　　F＝1　　'1989-09-02',0

　　　　　F＝2　　'89/09/02',0

　　　　　F＝3　　'89-09-02',0

A1.l　　作成した文字列データを書き込むアドレス

返り値

D0.l　　－1 のときエラー

A1.l　　文字列のエンド・アドレス（0のアドレス）

機能

バイナリ表現の日付データから日付を表わす文字列を作成します.

コール

```
moveq   #$5A,D0
move.l  #$07C50902,D1
lea     buf_add,A1
trap    #15
```

サンプル・プログラム

P.451　IOCSサンプル16

$5B _TIMEASC

引数

D1.l　　時刻データ（バイナリ）

　　$00_hh_mm_ss

　　　　$hh　　時バイナリ（0～23）

　　　　$mm　　分バイナリ（0～59）

　　　　$ss　　秒バイナリ（0～59）

A1.l　　作成した文字列データを書き込むアドレス

返り値

D0.l　　－1のときエラー

A1.l　　文字列のエンド・アドレス（0のアドレス）

　　文字列のフォーマットは '12:34:56',0

機能

バイナリ表現の時刻データから時刻を表わす文字列を作成します.

コール

```
        moveq    #$5B,D0
        move.l   #$00100231,D1
        lea      buf_add,A1
        trap     #15
```

サンプル・プログラム

P.451　IOCSサンプル16

$5C　_DAYASC

引数

D1.l　曜日（0〜6:0＝日曜, 6＝土曜）

A1.l　作成した文字列データを書き込むアドレス

返り値

A1.l　曜日を表わす文字列データ・アドレス

文字列のフォーマットは '水', 0

機能

曜日番号から曜日を表わす文字列データを作成します.

コール

```
        moveq    #$5C,D0
        moveq    #$1,D1
        lea      buf_add,A1
        trap     #15
```

サンプル・プログラム

P.451　IOCSサンプル16

$5D　_ALARMMOD

引数

D1.l　0　禁止

1　許可

2　現在の状態を調べる

返り値

D0.l　状態

0　禁止

1　許可

機能

アラームの禁止・許可を設定します.

コール

```
        moveq    #$5D,D0
        moveq    #$1,D1
        trap     #15
```

$5E　_ALARMSET

引数

D1.l　アラーム時間
　$0w_DD_HH_MM
　　$w　曜日カウンタ（0～6:0＝日曜，6＝土曜）
　　$DD　日BCD（1～$31）
　　$HH　時BCD（0～$23）
　　$MM　分BCD（0～$59）

D2.l　テレビ・コンピュータの電源をOFFにするまでの時間（分単位）
　0のときOFFにしない

A1.l　0　コンピュータON・テレビON
　－1　コンピュータON
　1～$3F　テレビコントロール
　上記以外は処理アドレスと見なされる（処理アドレスの先頭は$60：BRAにする）

返り値

D0.l　－1のときエラー

機能

アラーム時間と処理内容を設定します．

コール

```
moveq   #$5E,D0
move.l  #$03080800,D1
moveq   #$0,D2
lea     jobadd,A1
trap    #15
```

$5F　_ALARMGET

引数

なし

返り値

D0.l　処理アドレス

D1.l　アラーム時間
　$0w_DD_HH_MM
　　$w　曜日カウンタ（0～6：0＝日曜，6＝土曜）
　　$DD　日BCD（1～$31）
　　$HH　時BCD（0～$23）
　　$MM　分BCD（0～$59）

D2.l　テレビ・コンピュータの電源をOFFにするまでの時間（分単位）
　0のときOFFにしない

機能

アラーム時間と処理内容を調べます．

コール

```
moveq   #$5F,D0
trap    #15
```

$60　_ADPCMOUT

引数

D1.w　サンプリング周波数×256＋出力モード

サンプリング周波数

0	3.9KHz
1	5.2KHz
2	7.8KHz
3	10.4KHz
4	15.6KHz

出力モード

0	出力しない
1	左に出力
2	右に出力
3	左右に出力

D2.l　出力データのバイト数

A1.l　出力データ・アドレス

返り値

D0が破壊される

機能

ADPCMへデータを出力します．データの長さが$FF00バイト未満の場合はすぐにリターンしてきます．$FF00バイト以上の場合はすぐにリターンしません．

コール

```
moveq   #$60,D0
move.w  #4*256+3,D1
move.l  #$8000,D2
lea     pcmdata,A1
trap    #15
```

サンプル・プログラム

P.462　IOCSサンプル17

$61　_ADPCMINP

引数

D1.w　サンプリング周波数×256＋入力モード

サンプリング周波数

0	3.9KHz
1	5.2KHz
2	7.8KHz
3	10.4KHz
4	15.6KHz

入力モード

1	左から入力
2	右から入力
3	左右から入力

D2.l　データを入力するバイト数

A1.l　入力バッファアドレス

返り値

D0が破壊される

機能

ADPCMからデータを入力します．データの長さが$FF00バイト未満の場合はすぐにリターンしてきます．$FF00バイト以上の場合はすぐにリターンしません．

コール

```
moveq   #$61,D0
move.w  #4*256+3,D1
move.l  #$8000,D2
lea     pcmbuf,A1
trap    #15
```

サンプル・プログラム

P.452　IOCSサンプル17

$62　_ADPCMAOT

引数

D1.w　サンプリング周波数×256＋出力モード

サンプリング周波数

0	3.9KHz
1	5.2KHz
2	7.8KHz
3	10.4KHz
4	15.6KHz

出力モード

0	出力しない
1	左に出力
2	右に出力

3　　左右に出力

.1　出力データの個数

.1　出力データ・チェーン・テーブル・アドレス

```
0(A1).l      1番目のデータの先頭アドレス
4(A1).w      1番目のデータの長さ（1～$FFFF)
6(A1).l      2番目のデータの先頭アドレス
$A(A1).w     2番目のデータの長さ（1～$FFFF)
$C(A1).l     3番目のデータの先頭アドレス
$10(A1).w    3番目のデータの長さ（1～$FFFF)
  ※
```

返り値

D0が破壊される

機能

チェーン・テーブルによってADPCMへデータを出力します．

コール

```
        moveq   #$62,D0
        move.w  #3*256+3,D1
        move.l  #4,D2
        lea     pcmdata,A1
        trap    #15
```

例

```
        moveq   #$62,D0
        move.w  #4*256+3,D1
        move.l  #2,D2
        lea     pcmdata,A1
        trap    #15
          ※
pcmdata dc.l    data1add
        dc.w    data1len
        dc.l    data2add
        dc.w    data2len
```

この場合，data1addからのdata1lenバイトとdata2addからのdata2lenバイトが出力される．

$63　_ADPCMAIN

引数

D1.w　サンプリング周波数×256＋入力モード

サンプリング周波数

0　3.9KHz
1　5.2KHz
2　7.8KHz
3　10.4KHz
4　15.6KHz

入力モード

1　左から入力
2　右から入力
3　左右から入力

D2.l　入力データの個数

A1.l　入力データ・チェーン・テーブル・アドレス

0(A1).l	1番目のバッファの先頭アドレス
4(A1).w	1番目の入力データの長さ（1～$FFFF)
6(A1).l	2番目のバッファの先頭アドレス
$A(A1).w	2番目の入力データの長さ（1～$FFFF)
$C(A1).l	3番目のバッファの先頭アドレス
$10(A1).w	3番目の入力データの長さ（1～$FFFF)

※

返り値

D0が破壊される

機能

チェーン・テーブルによってADPCMからデータを入力します.

コール

```
moveq   #$63,D0
move.w  #2*256+3,D1
move.l  #$5,D2
lea     pcmdata,A1
trap    #15
```

$64　_ADPCMLOT

引数

D1.w　サンプリング周波数×256＋出力モード

サンプリング周波数

0	3.9KHz
1	5.2KHz
2	7.8KHz
3	10.4KHz
4	15.6KHz

出力モード

0	出力しない
1	左に出力
2	右に出力
3	左右に出力

A1.l　出力データ・アレイチェーン・テーブル・アドレス

0(A1).l	1番目のデータの先頭アドレス
4(A1).w	1番目のデータの長さ（1～$FFFF)
6(A1).l	2番目のデータのテーブル・アドレス
0(??1).l	2番目のデータの先頭アドレス （0ならテーブル終わり）
4(??1).w	2番目のデータの長さ（1～$FFFF)
6(??1).l	3番目のデータのテーブル・アドレス
0(??2).l	3番目のデータの先頭アドレス （0ならテーブル終わり）
4(??2).w	3番目のデータの長さ（1～$FFFF)
6(??2).l	4番目のデータのテーブル・アドレス

※

返り値

D0が破壊される

機能

アレイチェーン・テーブルによってADPCMへデータを出力します.

コール

```
moveq   #$64,D0
move.w  #4*256+3,D1
lea     pcmdata,A1
trap    #15
```

例

```
moveq   #$64,D0
move.w  #1*256+3,D1
lea     table1,A1
trap    #15
 ※
```

```
table1  dc.l    data1add
        dc.w    data1len
        dc.l    table2
          ※
table2  dc.l    data2add
        dc.w    data2len
        dc.l    table3
          ※
table3  dc.l    0
```

この場合，data1addからのdata1lenバイトとdata2addからのdata2lenバイトが出力される．

キーバッファに先行入力が溜まりすぎたら…

X68Kの先行入力バッファは64バイト用意されていますが，ここに先行入力が溜まりすぎるとパニックに陥る場合があります．

たとえば，適当でない文字が入力されたらビープ音が鳴るようになっているプログラム．このようなプログラムが走っている間に「適当でない文字」を山盛り先行入力してしまった場合は悲惨です．さらに，けたたましいビープ音を設定していた場合などはもっと悲惨です．

こんな場合は [SHIFT] ＋ [STOP] を押しましょう．先行入力バッファは見事にクリアされます．

この組み合わせで [STOP] キーを押した場合，[CTRL] ＋ [S] を押したのと同じことになり，プログラムを中断してしまうことはありません．

$65 _ADPCMLIN

引数

D1.w　サンプリング周波数×256＋入力モード

サンプリング周波数

0	3.9KHz
1	5.2KHz
2	7.8KHz
3	10.4KHz
4	15.6KHz

入力モード

1	左から入力
2	右から入力
3	左右から入力

A1.l　入力データ・アレイチェーン・テーブル・アドレス

0(A1).l	1番目のデータの先頭アドレス
4(A1).w	1番目のデータの長さ（1～$FFFF)
6(A1).l	2番目のデータのテーブル・アドレス
0(??1).l	2番目のデータの先頭アドレス (0ならテーブル終わり)
4(??1).w	2番目のデータの長さ（1～$FFFF)
6(??1).l	3番目のデータのテーブル・アドレス
0(??2).l	3番目のデータの先頭アドレス (0ならテーブル終わり)
4(??2).w	3番目のデータの長さ（1～$FFFF)
6(??2).l	4番目のデータのテーブル・アドレス

※

返り値

D0が破壊される

機能

アレイチェーン・テーブルによってADPCMへデータを入力します.

コール

```
moveq   #$65,D0
move.w  #4*256+3,D1
lea     pcmdata,A1
trap    #15
```

$66　_ADPCMSNS

引数

なし

返り値

D0.l	実行状態
0	何もしていない
2	出力中（IOCS $60)
4	入力中（IOCS $61)
$12	出力中（IOCS $62)
$14	入力中（IOCS $63)
$22	出力中（IOCS $64)
$24	入力中（IOCS $65)

機能

ADPCMの実行モードを調べます.

コール

```
        moveq   #$66,D0
        trap    #15
```

例

```
        moveq   #$66,D0
        trap    #15
```

IOCS $60による出力中の場合，D0.l＝2となる.

サンプル・プログラム

P.452　IOCSサンプル17

$67　_ADPCMMOD

引数

D1.l　制御コマンド

0	終了
1	中止
2	再開

返り値

D0が破壊される

機能

ADPCMの実行を制御します.

コール

```
        moveq   #$67,D0
        moveq   #0,D1
        trap    #15
```

$68　_OPMSET

引数

D1.b　　レジスタ・ナンバー

D2.b　　データ

返り値

D0が破壊される

機能

OPMのレジスタにデータを書き込みます．書き込める状態になるまで待ちます．

コール

```
moveq   #$68,D0
move.b  #$2F,D1
move.b  #$10,D2
trap    #15
```

サンプル・プログラム

P.452　IOCSサンプル18

$69　_OPMSNS

引数

なし

返り値

D0.b　　ステータス

bit 0　タイマーBオーバーフローのとき1になる

bit 1　タイマーAオーバーフローのとき1になる

bit 7　0ならばデータ書き込み可能

機能

OPMのステータスを読み込みます．

コール

```
moveq   #$69,D0
trap    #15
```

$6A　_OPMINTST

引数

A1.l　割り込み処理アドレス
　0のとき割り込み禁止

返り値

D0.l　0のとき割り込みが設定された
　それ以外の場合はすでに設定されている

機能

OPMによる割り込みの設定を行ないます．MFPに対してのみ設定するので，OPMのレジスタを別に設定しておく必要があります．割り込みを禁止する場合は，なるべくCPUを割り込み禁止状態にして実行してください．

コール

```
	moveq	#$6A,D0
	lea	intadd,A1
	trap	#15
```

$6B　_TIMERDST

引数

D1.w　単位時間×256＋カウンタ
　単位時間

1	1μs
2	2.5μs
3	4μs
4	12.5μs
5	16μs
6	25μs
7	50μs

A1.l　割り込み処理アドレス
　0のとき割り込み禁止

返り値

D0.l　0のとき割り込みが設定された
　それ以外の場合はすでに設定されている

機能

TIMER-Dによる割り込みを設定します．Human68Kのバージョン2.0Xではシステムが使うため設定できません．割り込みを禁止する場合は，なるべくCPUを割り込み禁止状態にして実行してください．

コール

```
	moveq	#$6B,D0
	move.w	#5*256+10,D1
	lea	intadd,A1
	trap	#15
```

$6C　_VDISPST

引数

D1.w　タイミング×256＋カウンタ
　　　タイミング
　　　0　　垂直帰線期間で割り込み
　　　1　　垂直表示期間で割り込み

A1.l　割り込み処理アドレス
　　　0のとき割り込み禁止

返り値

D0.l　0のとき割り込みが設定された
　　　それ以外の場合はすでに設定されている

機能

垂直同期による割り込みを設定します．割り込みを禁止する場合は，なるべくCPUを割り込み禁止状態にして実行してください．

コール

```
        moveq   #$6C,D0
        move.w  #$0*256+$10,D1
        lea     intadd,A1
        trap    #15
```

$6D　_CRTCRAS

引数

D1.w　ラスタ・ナンバー

A1.l　割り込み処理アドレス
　　　0のとき割り込み禁止

返り値

D0.l　0のとき割り込みが設定された
　　　それ以外の場合はすでに設定されている

機能

ラスタ操作による割り込みを設定します．指定したラスタを走査開始すると割り込みが発生します．割り込みを禁止する場合は，なるべくCPUを割り込み禁止状態にして実行してください．

コール

```
        moveq   #$6D,D0
        move.w  #$0,D1
        lea     intadd,A1
        trap    #15
```

サンプル・プログラム

P.454　IOCSサンプル19

$6E　_HSYNCST

引数

A1.l　割り込み処理アドレス
　　　0のとき割り込み禁止

返り値

D0.l　0のとき割り込みが設定された
　　　それ以外の場合はすでに設定されている

機能

水平同期による割り込みを設定します．水平同期信号の立ち下がり時に割り込みが発生します．割り込みを禁止する場合は，なるべくCPUを割り込み禁止状態にして実行してください．

コール

```
moveq   #$6E,D0
lea     intadd,A1
trap    #15
```

$6F　_PRNINITST

引数

A1.l　割り込み処理アドレス
　　　0のとき割り込み禁止

返り値

D0.l　0のとき割り込みが設定された
　　　それ以外の場合はすでに設定されている

機能

プリンタによる割り込みを設定します．割り込みを禁止する場合は，なるべくCPUを割り込み禁止状態にして実行してください．

コール

```
moveq   #$6F,D0
lea     intadd,A1
trap    #15
```

$70　_MS_INIT

引数

なし

返り値

D0が破壊される

機能

マウスを初期化します.

コール

```
        moveq   #$70,D0
        trap    #15
```

サンプル・プログラム

P.455　IOCSサンプル20

$71　_MS_CURON

引数

なし

返り値

D0が破壊される

機能

マウス・カーソルを表示します.

コール

```
        moveq   #$71,D0
        trap    #15
```

サンプル・プログラム

P.455　IOCSサンプル20

$72　_MS_CUROF

引数

なし

返り値

D0が破壊される

機能

マウス・カーソルを消します.

コール

```
        moveq   #$72,D0
        trap    #15
```

サンプル・プログラム

P.455　IOCSサンプル20

$73　_MS_STAT

引数

なし

返り値

D0.w	0	表示していない
	1	表示している

機能

マウス・カーソルが表示されているかどうかを調べます.

コール

```
        moveq   #$73,D0
        trap    #15
```

$74　_MS_GETDT

引数

なし

返り値

D0.l　マウスからのデータ

$XX_YY_LL_RR

XX	X方向移動量	
YY	Y方向移動量	
LL	$00	左ボタンOFF
	$FF	左ボタンON
RR	$00	右ボタンOFF
	$FF	右ボタンON

機能

マウスの移動量とボタンの状態を調べます.

コール

```
        moveq   #$74,D0
        trap    #15
```

サンプル・プログラム

P.455　IOCSサンプル20

$75 _MS_CURGT

引数

なし

返り値

D0.l マウス・カーソルの座標

$XXXX_YYYY

XXXX X座標

YYYY Y座標

機能

マウス・カーソルの座標を調べます.

コール

```
moveq   #$75,D0
trap    #15
```

サンプル・プログラム

P.455 IOCSサンプル20

$76 _MS_CURST

引数

D1.l マウス・カーソルの座標

$XXXX_YYYY

XXXX X座標

YYYY Y座標

返り値

D0.l −1のときエラー

機能

マウス・カーソルの座標を指定します.

コール

```
moveq   #$76,D0
move.l  #$01510012,D1
trap    #15
```

サンプル・プログラム

P.455 IOCSサンプル20

$77　_MS_LIMIT

引数

D1.l　マウス・カーソル移動範囲の座標（左上）

　$XXXX_YYYY

　　XXXX　X座標

　　YYYY　Y座標

D2.l　マウス・カーソル移動範囲の座標（右上）

　$XXXX_YYYY

　　XXXX　X座標

　　YYYY　Y座標

返り値

D0.l　−1のときエラー

機能

マウス・カーソルの移動範囲を設定します．

コール

```
moveq   #$77,D0
move.l  #$00000000,D1
move.l  #$00FF00FF,D2
trap    #15
```

サンプル・プログラム

P.455　IOCSサンプル20

$78　_MS_OFFTM

引数

D1.w　調べるボタン

　0　左ボタン

　−1　右ボタン

D2.w　待時間

　0　離すまで待つ

　上記以外の場合，待時間の最大値

返り値

D0.w　0　ドラッグ中

　−1　待時間の最大値を越えた

　上記以外の場合，待時間

機能

マウスのボタンを離すまでの時間を調べます．

コール

```
moveq   #$78,D0
move.w  #-1,D1
move.w  #$0,D2
trap    #15
```

$79 _MS_ONTM

引数

D1.w　調べるボタン
　0　　左ボタン
　−1　　右ボタン
D2.w　待時間
　0　　離すまで待つ
　上記以外の場合，待時間の最大値

返り値

D0.w　0　　ドラッグ中
　−1　　待時間の最大値を越えた
　上記以外の場合，待時間

機能

マウスのボタンを押すまでの時間を調べます．

コール

```
        moveq   #$79,D0
        move.w  #-1,D1
        move.w  #$0,D2
        trap    #15
```

もうひとつのCコンパイラ・GNU C

X68000にはXCという，いろんな意味で「強力な」Cコンパイラが用意されています．X-BASICからCコンバートして，それをコンパイルしてしまうという大技や，X68000の機能のほとんどをサポートしてしまう大量のライブラリなど，これはこれで充分使えるCコンパイラです．

しかし，世の中にはつねにもっと上を望む人々というのはいるもので，そういう人たちが産み出してしまったのがGNU Cコンパイラ・X68000版です．

このコンパイラはXCよりも整理された，速いコードを生成してくれます．が，その代償として，とにかくなにかと豪快なやつで，コンパイラ自体も，作業に必要なメモリも，XCよりも巨大で，最低1MBのRAM増設は必要です．まともに使おうと思ったらハードディスクも必要でしょう（これはXCも同じですが）．

パワー・ユーザーご用達のGNU C．興味のある方は，パソコン通信を通じてあちこち捜してみるとよいでしょう．GNU Cはフリーウェアですから，原則として無料でわけてもらうことができます．そのかわり，原作者，移植者の方への感謝の気持ち（できれば言葉も）を忘れなく．

なお，GNU CでもXCのライブラリが必要です．XCをお持ちでない方は使えませんので念のため．

$7A　_MS_PATST

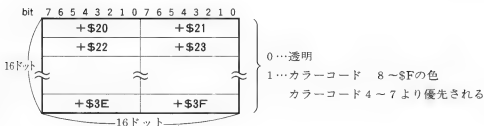
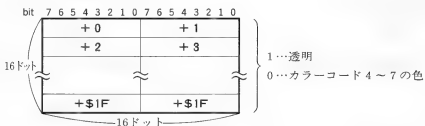
引数

D1.w　カーソルナンバー

A1.l　パターン・データ・アドレス

0(A1).w	パターンの左端X座標からマウスX座標までの距離
2(A1).w	パターンの上端Y座標からマウスY座標までの距離
4(A1).b-	パターン・データ（0）
$24(A1).b-	パターン・データ（1）

マウスカーソルデーターパターン

返り値

D0が破壊される

機能

マウス・カーソルのパターンを定義します。

コール

```
moveq   #$7A,D0
move.w  #$2,D1
lea     patdata,A1
trap    #15
```

サンプル・プログラム

P.455　IOCSサンプル20

$7B　_MS_SEL

引数

D1.w　カーソルナンバー

返り値

D0が破壊される

機能

マウス・カーソルのパターンを選びます．

コール

```
moveq   #$7B,D0
move.w  #$1,D1
trap    #15
```

$7C　_MS_SEL2

引数

A1.1　カーソル・ナンバーテーブル・アドレス

0(A1).w	カーソルナンバー1
2(A1).w	カーソルナンバー2
4(A1).w	カーソルナンバー3
※	
??(A1).w	−1（エンドマーク）

返り値

D0が破壊される

機能

マウス・カーソル・パターンを複数個表示させることによってアニメーション表示します．

コール

```
moveq   #$7C,D0
lea     numtable,A1
trap    #15
```

サンプル・プログラム

P.455　IOCSサンプル20

$7D　_SKEY_MOD

引数

D1.l　0　ソフトキーボード消去
　　　1　ソフトキーボード表示
　　　2　ソフトキーボードの表示状態を調べる
　　　−1　ソフト・キーボード自動制御（マウス右ボタンによる制御）

D2.l　表示座標（D1.l=1）
　　　$XXXX_YYYY
　　　　XXXX　X座標
　　　　YYYY　Y座標

返り値

D0.l　ソフト・キーボードの表示状態（D1.l＝2）
　　　0　表示していない
　　　1　表示している

機能

ソフト・キーボードの制御を行ないます.

コール

```
moveq   #$7D,D0
move.l  #$1,D1
move.l  #$00100010,D2
trap    #15
```

$7E　_DENSNS

引数

なし

返り値

D0が破壊される

機能

電卓による入力を調べます. 結果はキー入力として返ります.

コール

```
moveq   #$7E,D0
trap    #15
```

$7F　_ONTIME

引数

なし

返り値

D0.l　経過時間（単位は1/100秒）
　$0から$83D5FF

D1.l　経過時間（単位は日）
　$0から$FFFF

機能

起動してからの経過時間を調べます．

コール

```
        moveq   #$7F,D0
        trap    #15
```

$80　_B_INTVCS

引数

D1.w　ベクタナンバー
　0～$FF　割り込みベクタ（割り込みルーチンの最後は「RTE」）
　$100～$1FF　IOCSコールベクタ（処理ルーチンの最後は「RTS」）

A1.l　処理アドレス
　（このアドレスにジャンプする時点でスーパーバイザ・モードになっている）

返り値

D0.l　設定前の処理アドレス

機能

各種ベクタ・テーブルを書きかえます．X68000は通常さまざまなタイマー割り込みなどがかかっています．書き換える前に処理ルーチンを用意してください．

コール

```
        moveq   #$80,D0
        move.w  #$120,D1
        lea     intadd,A1
        trap    #15
```

$81　_B_SUPER

引数

A1.l　0　　　　　ユーザー・モードからスーパーバイザ・モードに切り替える
　　　以前のSSP　　スーパーバイザ・モードからユーザー・モードに切り替える

返り値

D0.l　設定時のSSP　（ユーザー・モード→スーパーバイザ・モード）
　　　0　　　　　　（スーパーバイザ・モード→ユーザー・モード）
　　　−1のときエラー

機能

スーパーバイザ・モード，ユーザー・モードの切り替えを行ないます．ユーザー・モードではメインRAMの0番地から指定された番地までと，$C00000番地以降はアクセスできません．

コール

```
moveq   #$81,D0
clr.l   A1
trap    #15
```

$82　_B_BPEEK

引数

A1.l　読み込みアドレス

返り値

D0.b　読み込みデータ
A1.l　次のアドレス

機能

指定アドレスから1バイト読み込みます．ユーザー・モードのときに，スーパーバイザ・エリアをアクセスする場合に使います．

コール

```
moveq   #$82,D0
lea     work,A1
trap    #15
```

$83　_B_WPEEK

引数

A1.l　読み込みアドレス（偶数番地）

返り値

D0.w　読み込みデータ

A1.l　次のアドレス

機能

指定アドレスから１ワード読み込みます．ユーザー・モードのときに，スーパーバイザ・エリアをアクセスする場合に使います．

コール

```
moveq	#$83,D0
lea	work,A1
trap	#15
```

$84　_B_LPEEK

引数

A1.l　読み込みアドレス（偶数番地）

返り値

D0.l　読み込みデータ

A1.l　次のアドレス

機能

指定アドレスから１ロングワード読み込みます．ユーザー・モードのときに，スーパーバイザ・エリアをアクセスする場合に使います．

コール

```
moveq	#$84,D0
lea	work,A1
trap	#15
```

$85 _B_MEMSTR

引数

A1.l　読み込みアドレス

A2.l　バッファアドレス

D1.l　読み込むデータのバイト数－1

返り値

D0が破壊される

A1.l　次のアドレス

A2.l　次のバッファアドレス

機能

指定アドレスから複数バイトのデータを読み込みます．ユーザー・モードのときに，スーパーバイザ・エリアをアクセスする場合に使います．

コール

```
moveq   #$85,D0
lea     work,A1
lea     bufadd,A2
move.l  #$1000-1,D1
trap    #15
```

$86 _B_BPOKE

引数

D1.b　書き込みデータ

A1.l　書き込みアドレス

返り値

D0が破壊される

A1.l　次のアドレス

機能

指定アドレスに1バイト・データを書き込みます．ユーザー・モードのときに，スーパーバイザ・エリアをアクセスする場合に使います．

コール

```
moveq   #$86,D0
move.b  #$12,D1
lea     work,A1
trap    #15
```

$87　_B_WPOKE

引数

D1.w　書き込みデータ

A1.l　書き込みアドレス（偶数番地）

返り値

D0が破壊される

A1.l　次のアドレス

機能

指定アドレスに１ワード・データを書き込みます．ユーザー・モードのときに，スーパーバイザ・エリアをアクセスする場合に使います．

コール

```
        moveq   #$87,D0
        move.w  #$1234,D1
        lea     work,A1
        trap    #15
```

$88　_B_LPOKE

引数

D1.l　書き込みデータ

A1.l　書き込みアドレス（偶数番地）

返り値

D0が破壊される

A1.l　次のアドレス

機能

指定アドレスに１ロングワード・データを書き込みます．ユーザーモードのときに，スーパーバイザ・エリアをアクセスする場合に使います．

コール

```
        moveq   #$88,D0
        move.l  #$12345678,D1
        lea     work,A1
        trap    #15
```

$89　_B_MEMSET

引数

A1.l　　書き込みアドレス

A2.l　　データ・アドレス

D1.l　　書き込むデータのバイト数−1

返り値

D0が破壊される

A1.l　　次のアドレス

A2.l　　次のデータ・アドレス

機能

指定アドレスに複数バイトのデータを書き込みます．ユーザーモードのときに，スーパーバイザ・エリアをアクセスする場合に使います．

コール

```
	moveq	#$89,D0
	lea	work,A1
	lea	dataadd,A2
	move.l	#$400−1,D1
	trap	#15
```

$8A　_DMAMOVE

引数

D1.b　転送モード

bit 7	0	(A1) から (A2) へ転送する
	1	(A2) から (A1) へ転送する
bit 3〜2	%00	A1を固定する
	%01	A1を増やす（＋1）
	%10	A1を減らす（−1）
bit 1〜0	%00	A2を固定する
	%01	A2を増やす（＋1）
	%10	A2を減らす（−1）

D2.l　転送するバイト数

A1.l　転送アドレス1

A2.l　転送アドレス2

返り値

D0が破壊される

機能

DMA転送を行ないます．データの長さが$FF00バイト未満の場合はすぐにリターンしてきます．$FF00バイト以上の場合はすぐにリターンしません．

コール

```
moveq   #$8A,D0
move.b  #%00000101,D1
move.l  #$800,D2
lea     dataadd,A1
lea     work,A2
trap    #15
```

サンプル・プログラム

P.457　IOCSサンプル21

$8B　_DMAMOV_A

引数

D1.b　転送モード

bit 7	0	（アドレス1）から（アドレス2）へ転送する
	1	（アドレス2）から（アドレス1）へ転送する
bit 3〜2	%00	アドレス1を固定する
	%01	アドレス1を増やす（+1）
	%10	アドレス1を減らす（−1）
bit 1〜0	%00	アドレス2を固定する
	%01	アドレス2を増やす（+1）
	%10	アドレス2を減らす（−1）

D2.l　転送データ・チェーン・テーブルの個数

A1.l　転送データ・チェーン・テーブル・アドレス

0(A1).l	転送アドレス1（1）
4(A1).w	転送データ・バイト数1（1〜$FFFF）
6(A1).l	転送アドレス1（2）
$A(A1).w	転送データ・バイト数2（1〜$FFFF）
$C(A1).l	転送アドレス1（3）
$10(A1).w	転送データ・バイト数3（1〜$FFFF）
※	

A2.l　転送アドレス2

返り値

D0が破壊される

機能

チェーン・テーブルによるDMA転送を行ないます.

コール

```
moveq   #$8B,D0
move.b  #%00000101,D1
move.l  #2,D2
lea     table,A1
lea     work,A2
trap    #15
```

例

```
moveq   #$8B,D0                 table   dc.l    dat1add
move.b  #%0101,D1                       dc.w    dat1len
move.l  #3,D2                           dc.l    dat2add
lea     table,A1                        dc.w    dat2len
lea     datadd,A2                       dc.l    dat3add
trap    #15                             dc.w    dat3len
   :
```

この場合，dat1addからのdat1lenバイト，dat2addからのdat2lenバイト，dat3addからのdat3lenバイトがこの順番で連続してdatadd以降に転送される.

$8C　_DMAMOV_L

引数

D1.b　転送モード

bit 7	0	(アドレス1)から(アドレス2)へ転送する
	1	(アドレス2)から(アドレス1)へ転送する
bit 3〜2	%00	アドレス1を固定する
	%01	アドレス1を増やす(+1)
	%10	アドレス1を減らす(−1)
bit 1〜0	%00	アドレス2を固定する
	%01	アドレス2を増やす(+1)
	%10	アドレス2を減らす(−1)

A1.l　転送データ・アレイチェーン・テーブル・アドレス

0(A1).l	転送アドレス1(1)
4(A1).w	転送データ・バイト数1(1〜$FFFF)
6(A1).l	次のテーブル・アドレス
0(?1).l	転送アドレス1(2)
4(?1).w	転送データ・バイト数2(1〜$FFFF)
6(?1).l	次のテーブル・アドレス(0のときテーブルエンド)
0(?2).l	転送アドレス1(3)
4(?2).w	転送データ・バイト数3(1〜$FFFF)
6(?2).l	次のテーブル・アドレス(0のときテーブルエンド)
※	

A2.l　転送アドレス2

返り値

D0が破壊される

機能

アレイチェーンによるDMA転送を行ないます.

コール

```
        moveq   #$8C,D0
        move.b  #%00000101,D1
        lea     table,A1
        lea     work,A2
        trap    #15
```

例

```
        moveq   #$8C,D0
        move.b  #%00000101,D1
        lea     table1,A1
        lea     datadd,A2
        trap    #15
          ※
table1  dc.l    dat1add
        dc.w    dat1len
        dc.l    table2
```

```
table2  dc.l    dat2add
        dc.w    dat2len
        dc.l    table3
table3  dc.l    dat3add
        dc.w    dat3len
        dc.l    0
```

この場合，dat1addからのdat1lenバイト，dat2addからのdat2lenバイト，dat3addからのdat3lenバイトがこの順番で連続してdatadd以降に転送される．

$8D　_DMAMODE

引数

なし

返り値

D0.l　DMA実行モード

0	何もしていない
$8A	IOCS $8A実行中
$8B	IOCS $8B実行中
$8C	IOCS $8C実行中

機能

DMAの実行モードを調べます．

コール

```
        moveq   #$8D,D0
        trap    #15
```

Amigaへのラブコール

とにかく，究極の68000パソコンと言えば，誰が何と言おうとAmigaです(ちなみに，Macは至高の68000パソコン)．

なにしろ，7.6MHzの68000で，マルチウィンドウでマルチタスクのOSが動いているというとんでもない代物．ゲームだって，グラフィックスだって，音楽だって驚異的なクオリティでこなします．世間ではゲーム機だと思われているかもしれませんが，とんでもない．Amigaで作ったTV番組のタイトル・グラフィックスなんかは週に何度もお茶の間で目にできるはずです．

ハードウェアの能力なら，我等がX68000だって負けてはいません．グラフィックの分解能，色数，特殊効果機能，スプライト機能，どれをとってもAmigaにひけはとりません(本当は部分的にひけをとっているのですが…)．しかし，ソフトウェアではまだまだAmigaに及ばないのが実情です．

Amigaプログラマーたちの恐ろしいまでのパワーを見習って，私たちもX68000を究極のパソコンにすべく，日夜努力しようじゃありませんか．

$8E _BOOTINF

引数

なし

返り値

	起動情報	
下位24ビット	$80～$83	ハードディスクから起動
	$90～$93	2HDフロッピーディスクから起動
	$ED0000～$ED3FFE	SRAMから起動
	上記以外	ROMから起動
上位8ビット	$00	パワースイッチによって起動
	$01	外部スイッチによって起動
	$02	タイマーによって起動

機能

起動情報を調べます．

コール

```
moveq   #$8E,D0
trap    #15
```

例

```
moveq   #$8E,D0
trap    #15
```

このとき，D0.1＝$00000090となった場合は，パワースイッチによって2HDフロッピーディスク・ドライブ0から起動している．

$8F _ROMVER

引数

なし

返り値

D0.1　バージョン・データ

$VV_YY_MM_DD

VV	バージョン（BCD表現）
YY	年（BCD表現）
MM	月（BCD表現）
DD	日（BCD表現）

機能

ROMのバージョンと作成年月日を調べます．

コール

```
moveq   #$8F,D0
trap    #15
```

$90 _G_CLR_ON

引数

なし

返り値

D0が破壊される

機能

グラフィック画面をクリアして表示モードにします．パレットは標準に戻ります．

コール

```
        moveq   #$90,D0
        trap    #15
```

サンプル・プログラム

P.458 IOCSサンプル22

$91 _CRTMOD2

引数

D1.b	グラフィック画面モード	
0	16色４画面	512×512
1	256色２画面	512×512
3	65536色１画面	512×512
4	16色１画面	1024×1024
−1	現在のモードを調べる	

返り値

D0.1 グラフィック画面モード（D1.b＝−1）

機能

グラフィック画面モードを設定します．

コール

```
        moveq   #$91,D0
        move.b  #$4,D1
        trap    #15
```

$92 _CRTPRI

引数

D1.w　プライオリティデータ

bit $D～$C　スプライト画面の優先順位（0～2）
bit $B～$A　テキスト画面の優先順位（0～2）
bit 9～8　グラフィック画面の優先順位（0～2）
以上は0がいちばん優先順位が高く，2がいちばん低い
bit 7～0　グラフィック・ページ間の優先順位
グラフィック画面4ページのとき
bit 7～6　1番優先順位の高いページナンバー（0～3）
bit 5～4　2番目に優先順位の高いページナンバー（0～3）
bit 3～2　3番目に優先順位の高いページナンバー（0～3）
bit 1～0　1番優先順位の低いページナンバー（0～3）
グラフィック画面2ページのとき
bit 7～0　%1110_0100のときページ0を優先表示
%0100_1110のときページ1を優先表示
−1のとき現在のプライオリティを調べる

返り値

D0.l　プライオリティデータ（D1.w＝−1）

機能

プライオリティの設定を行ないます。

コール

```
moveq   #$92,D0
move.w  #%01001001110010,D1
trap    #15
```

$93 _G_SET_ON2

引数

D1.w　画面表示切り替え・特殊モード切り替えデータ

bit $E　1のとき，テキスト・パレット0とグラフィック画面とのあいだで半透明表示

bit $D　1のとき，テレビ&ビデオ画面と最も表示優先順位が高いグラフィック画面の指定されたエリアとの間で半透明表示

bit $C　1のとき，半透明機能と特殊プライオリティ機能が有効になる

bit $B　1のとき半透明モード，0のとき特殊プライオリティ・モード

bit $A　1のとき，最も表示優先順位の高いグラフィックVRAMのデータで半透明と特殊プライオリティを機能させるエリアを指定する

bit 9　1のとき，グラフィック画面（優先順位が1番と2番）どうしで半透明表示

bit 8　1のとき，テキスト画面と最も表示優先順位の高いグラフィック画面の指定されたエリアとの間で半透明表示

bit 6　スプライト画面表示（1のとき表示）

bit 5　テキスト画面表示（1のとき表示）

bit 4〜0 グラフィック画面表示オン・オフ

1024×1024グラフィック画面のとき

bit 4　1のとき表示

512×512グラフィック画面のとき

4ページのとき（1のとき表示）

bit 3　1番優先順位が低い画面表示

bit 2　3番目に優先順位が高い画面表示

bit 1　2番目に優先順位が高い画面表示

bit 0　1番優先順位が高い画面表示

2ページのとき（%11のとき表示）

bit 3〜2 優先順位が高い画面表示

bit 1〜0 優先順位が低い画面表示

1ページのとき（%1111のとき表示）

bit 3〜0 画面表示

返り値

D0が破壊される

機能

画面表示と特殊モードの設定を行ないます．

コール

```
moveq   #$93,D0
move.w  #%00011110_01101011,D1
trap    #15
```

$94 _GPALET

引数

D1.w　パレットコード

D2.l　カラーコード

　　　−1のときは現在のカラーコードを調べる

返り値

D0.l　現在のカラーコード（D2.l＝−1）

機能

グラフィック・パレットを設定します（なお，1987年3月18日バージョンのROMの場合，256×256ドット・モードでは正常に動作しません）.

コール

```
moveq   #$94,D0
move.w  #$34,D1
move.l  #$1234,D2
trap    #15
```

$95 _GPALET2

引数

D1.l　パレットコード

返り値

D0が破壊される

機能

グラフィック・カラーコードを設定します．IOCS $9A,$9B,$9Cで使用されます.

コール

```
moveq   #$95,D0
move.l  #$1234,D1
trap    #15
```

サンプル・プログラム

P.459　IOCSサンプル23

$96 _APAGE2

引数

D1.l　読み書きするグラフィック・ページナンバー（0〜3）

返り値

D0が破壊される

機能

グラフィック・ページを設定します．IOCSで使用されます（IOCS $B1と共通）.

コール

```
moveq   #$96,D0
move.l  #1,D1
trap    #15
```

$97 G_READ

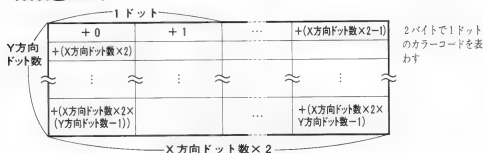
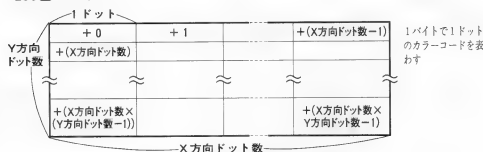
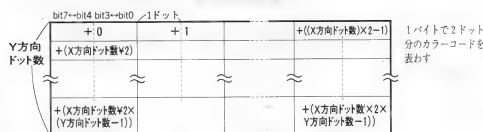
引数

D1.w　Xドット座標

D2.w　Yドット座標

A1.l　バッファアドレス

(A1).w	読み出すX方向ドット数
2(A1).w	読み出すY方向ドット数
4(A1).w	カラーモード・バッファ
	$F　16色モード
	$FF　256色モード
	$FFFF　65536色モード
6(A1).b-	バッファ
	16色モードのときは1バイトに2ドット分セットされる

グラフィックVRAM読み書きデーターフォーマット

65536色モード

2バイトで1ドットのカラーコードを表わす

256色モード

1バイトで1ドットのカラーコードを表わす

16色モード(X方向ドット数が偶数の場合)

1バイトで2ドット分のカラーコードを表わす

16色モード（X方向のドット数が奇数の場合）

基本的に偶数のときと同じ。ただし，右端に余る4ビット（半バイト＝1ドット）に次のラインの左端のドットのカラーコードが対応し，以降同様に詰めていく．

(例)

8	1	0	0	A
D	B	4	3	2
1	0	9	8	9

このパターンは次のようになる．

+0	+1	+2	+3	+4	+5	+6	+7
$81	$00	$AD	$B4	$32	$10	$98	$90

↑ 最後の0が余り

返り値

D0が破壊される

機能

グラフィック画面からカラーパターンを読み出します．読み出しデータは4(A1).wと6(A1).b 以降にセットされます．

コール

```
moveq    #$97,D0
move.w   #$121,D1
move.w   #$21,D2
lea      bufadd,A1
trap     #15
```

$98 _G_W_COLPAT

引数

D1.w　Xドット座標

D2.w　Yドット座標

D3.w　書き込まないパレットコード

A1.l　データ・アドレス

(A1).w　書き込むX方向ドット数

2(A1).w　書き込むY方向ドット数

4(A1).w　カラーモード

$F　16色モード

$FF　256色モード

$FFFF　65536色モード

6(A1).b～　書き込むデータ

16色モードのときは1バイトに2ドット分セットする

グラフィックVRAM読み書きデーターフォーマット

65536色モード

(1ドット / Y方向ドット数 / X方向ドット数×2)

+0	+1	…	+(X方向ドット数×2−1)
+(X方向ドット数×2)			
⋮	⋮		⋮
+(X方向ドット数×2×(Y方向ドット数−1))		…	+(X方向ドット数×2×Y方向ドット数−1)

2バイトで1ドットのカラーコードを表わす

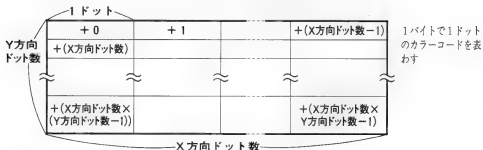
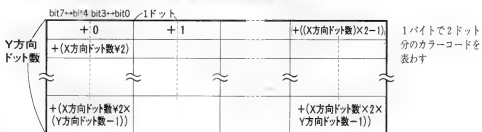
256色モード

1バイトで1ドットのカラーコードを表わす

16色モード(X方向ドット数が偶数の場合)

(bit7↔bit4 bit3↔bit0 / 1ドット / Y方向ドット数)

+0	+1		+((X方向ドット数)×2−1)
+(X方向ドット数¥2)			
+(X方向ドット数¥2×(Y方向ドット数−1))			+(X方向ドット数×2×Y方向ドット数−1))

1バイトで2ドット分のカラーコードを表わす

○色モード（X方向のドット数が奇数の場合）

基本的に偶数のときと同じ。ただし，右端に余る4ビット（半バイト＝1ドット）に次ラインの左端のドットのカラーコードが対応し，以降同様に詰めていく．

8	1	0	0	A
D	B	4	3	2
1	0	9	8	9

このパターンは次のようになる．

＋0	＋1	＋2	＋3	＋4	＋5	＋6	＋7
$81	$00	$AD	$B4	$32	$10	$98	$90

↑最後の0が余り

返り値

D0が破壊される

機能

グラフィック画面にカラーパターンを書き込みます．D3レジスタで指定したパレットコードは書き込まれません（そのドットには書き込みを行なわない）．

コール

```
moveq   #$98,D0
move.w  #$41,D1
move.w  #$18F,D2
move.w  #$10,D3
lea     datadd,A1
trap    #15
```

サンプル・プログラム

P.455　IOCSサンプル20

$99

引数

D1.w　　Xドット座標

D2.w　　Yドット座標

A1.l　　データ・アドレス

(A1).w	書き込むX方向ドット数		
2(A1).w	書き込むY方向ドット数		
4(A1).w	カラーモード		
	$F	16色モード	
	$FF	256色モード	
	$FFFF	65536色モード	
6(A1).b-	書き込むデータ		
	16色モードのときは1バイトに2ドット分セットする		

グラフィックVRAM読み書きデーターフォーマット

65536色モード

256色モード

16色モード(X方向ドット数が偶数の場合)

16色モード（X方向のドット数が奇数の場合）

基本的に偶数のときと同じ．ただし，右端に余る4ビット（半バイト＝1ドット）に次のラインの左端のドットのカラーコードが対応し，以降同様に詰めていく．

(例)

8	1	0	0	A
D	B	4	3	2
1	0	9	8	9

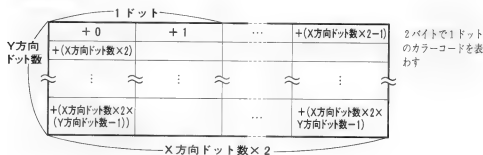
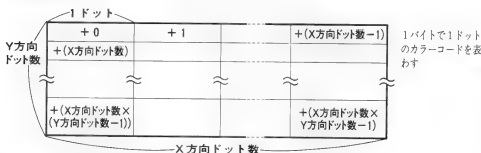
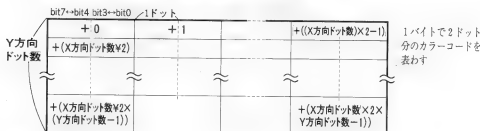
このパターンは次のようになる．

+0	+1	+2	+3	+4	+5	+6	+7
$81	$00	$AD	$B4	$32	$10	$98	$90

↑ 最後の0が余り

返り値

D0が破壊される

機能

グラフィック画面にカラーパターンを書き込みます．IOCS $98と違い，書き込む範囲のドットすべてに書き込みます．

コール

```
        moveq   #$98,D0
        move.w  #$41,D1
        move.w  #$18F,D2
        lea     datadd,A1
        trap    #15
```

サンプル・プログラム

P.458　IOCSサンプル22

$9A _G_W_BITPAT

引数

D1.w　　Xドット座標

D2.w　　Yドット座標

A1.l　　データ・アドレス

　　　　(A1).w　書き込むX方向ドット数

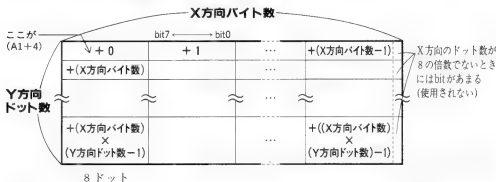
　　　　2(A1).w 書き込むY方向ドット数

　　　　4(A1).b 書き込むデータ

グラフィックVRAMビットパターン書き込みデーターフォーマット

返り値

D0が破壊される

機能

　グラフィック画面にビット・パターンを書き込みます．ビットが1の点はグラフィック・カラーコード（IOCS $95で設定）が書き込まれ，0の点には書き込みません．

コール

```
        moveq   #$9A,D0
        move.w  #$41,D1
        move.w  #$10,D2
        lea     datadd,A1
        trap    #15
```

サンプル・プログラム

P,459　IOCSサンプル23

$9B _G_W_BITPAT

引数

D1.w　　Xドット座標
D2.w　　Yドット座標
D3.w　　バック・カラーコード
A1.l　　データ・アドレス
　　　　(A1).w　書き込むX方向ドット数
　　　　2(A1).w 書き込むY方向ドット数
　　　　4(A1).b 書き込むデータ

返り値

D0が破壊される

機能

グラフィック画面にビット・パターンを書き込みます．ビットが1の点はグラフィック・カラーコード（IOCS $95で設定）が書き込まれ，0の点にはバックカラーコードが書き込まれます．

コール

```
        moveq   #$9B,D0
        move.w  #$41,D1
        move.w  #$10,D2
        move.w  #0,D3
        lea     datadd,A1
        trap    #15
```

サンプル・プログラム

P.459　IOCSサンプル23

$9C _G_W_WIDE_BIT

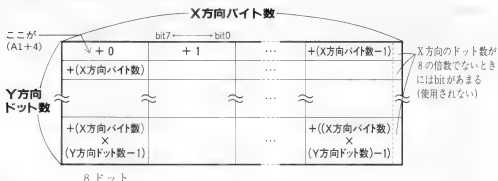
引数

D1.w　　Xドット座標
D2.w　　Yドット座標
D3.w　　X方向拡大倍率
D4.w　　Y方向拡大倍率
A1.l　　データ・アドレス
　　　　(A1).w　書き込むX方向ドット数
　　　　2(A1).w 書き込むY方向ドット数
　　　　4(A1).b 書き込むデータ

グラフィックVRAMビットパターン書き込みデーターフォーマット

返り値

D0が破壊される

機能

グラフィック画面にビット・パターンを拡大して書き込みます．ビットが1の点はグラフィック・カラーコード（IOCS $95で設定）が書き込まれ，0の点には書き込みません．

コール

```
        moveq   #$9C,D0
        move.w  #$40,D1
        move.w  #$10,D2
        move.w  #2,D3
        move.w  #4,D4
        lea     datadd,A1
        trap    #15
```

サンプル・プログラム

P.459　IOCSサンプル23

$A0　_SFTJIS

引数

D1.w　シフトJIS漢字コード

返り値

D0.l　JIS漢字コード
漢字コードが正しくない場合はD0.l＝$FFFF2228が返る

D1.l　D0.lと同じ

機能

シフトJIS漢字コードをJIS漢字コードに変換します.

コール

```
moveq   #$A0,D0
move.w  #'漢',D1
trap    #15
```

$A1　_JISSFT

引数

D1.w　JIS漢字コード

返り値

D0.l　シフトJIS漢字コード
漢字コードが正しくない場合はD0.l＝$FFFF81A6が返る

D1.l　D0.lと同じ

機能

JIS漢字コードをシフトJIS漢字コードに変換します.

コール

```
moveq   #$A1,D0
move.w  #$3B54,D1
trap    #15
```

$A2　_AKCONV

引数

D1.l　上位16ビット
　　0　平仮名にする
　　1　片仮名にする
　下位16ビット
　ANKコード（$20～$7E, $A1～$DF）

返り値

D0.l　シフトJIS漢字コード
　ANKコードが正しくない場合はD0.l＝$FFFF81A6が返る

機能

ANKコードを対応した全角シフトJISコードに変換します．

コール

```
moveq   #$A2,D0
move.l  #$10000+'ｱ',D1
trap    #15
```

例

```
moveq   #$A2,D0
move.l  #'ｱ',D1
trap    #15
```

この場合，D0.l＝'あ'となる．

$A3　_RMACNV

引数

D1.b　ローマ字の文字（アルファベット）
A1.l　変換用ワーク・アドレス（初めて変換する場合は先頭に0をセットしておく）
A2.l　変換結果を格納するアドレス

返り値

D0.l　ステータス
　$0　変換途中でまだ返す文字列がない
　－1　変換できない文字が送られた
　上記以外のときは変換結果文字数が返る
変換結果が出た場合は，D0.lに加えて変換結果も返る
　(A2).b　変換結果（ANK文字）
　1(A2).b　0

機能

ローマ字仮名変換を行ないます．アルファベットはそれぞれ対応した仮名に変換されます．変換途中の場合は，ワーク・アドレスにデータが保存され，変換結果は返りません．

コール

```
moveq   #$A3,D0
move.b  #'M',D1
lea     work,A1
lea     string,A2
clr.b   (A1)
trap    #15
```

$A4　_DAKJOB

引数

A1.1　全角文字列の最終アドレス＋1（内容は0にする）

返り値

D0.1　文字列の増えたバイト数

　　0　最後の文字が濁点付き文字になった

　　2　濁点を追加した

A1.1　全角文字列の最終アドレス＋1（内容は0）

機能

全角文字列の濁点処理を行ないます．最後の全角文字に濁点がつく場合は（か・さ・た・は…）その文字が，濁点付き文字に変換されます（が・ざ・だ・ば…）．濁点を付けられない文字の場合は単に全角の濁点が付け足されます．

コール

```
        moveq   #$A4,D0
        lea     string,A1
        trap    #15
```

例

```
        moveq   #$A4,D0
        lea     str,A1
        trap    #15
          ※
        dc.b    'コンピュータ'
str     dc.b    0
```

この場合，'タ'の文字が'ダ'に変換される．

$A5　_HANJOB

引数

A1.1　全角文字列の最終アドレス＋1（内容は0にする）

返り値

D0.1　文字列の増えたバイト数

　0　最後の文字が半濁点付き文字になった

　2　半濁点を追加した

A1.1　全角文字列の最終アドレス＋1（内容は0）

機能

全角文字列の半濁点処理を行ないます．最後の全角文字に半濁点がつく場合は（は・ひ…）その文字が，半濁点付き文字に変換されます（ぱ・ぴ…）．半濁点を付けられない文字の場合は単に全角の半濁点が付け足されます．

コール

```
        moveq   #$A5,D0
        lea     string,A1
        trap    #15
```

例

```
        moveq   #$A5,D0
        lea     str,A1
        trap    #15
          ※
        dc.b    'はひふへほ'
str     dc.b    0
```

この場合，'ほ'の文字が'ぽ'に変換される．

$AE　_OS_CURON

引数

なし

返り値

D0が破壊される

機能

カーソル表示モードにします．

コール

```
        moveq   #$AE,D0
        trap    #15
```

$AF　　_OS_CUROF

引数

なし

返り値

D0が破壊される

機能

カーソルを表示しないモードにします.

コール

```
        moveq   #$AE,D0
        trap    #15
```

$B1　　_APAGE

引数

D1.b　　書き込みページナンバー（0～3）
　　　　－1のとき現在の書き込みページを調べる

返り値

D0.l　　ステータス（D1.b<>－1）
　　　　0　　　正常終了
　　　　－1　　グラフィック画面が使えない
　　　　－2　　ページナンバーがおかしい
　　　　－3　　指定されたページナンバーは現在の画面モードでは使えない
　　　　ページナンバー（D1.b＝－1）

機能

グラフィック画面の読み込み，書き込みページを設定します.

コール

```
        moveq   #$B1,D0
        move.b  #2,D1
        trap    #15
```

サンプル・プログラム

P.460　IOCSサンプル24

$B2 _VPAGE

引数

D1.b　表示ページ指定データ

bit 0　ページ0（1のとき表示）
bit 1　ページ1（1のとき表示）
bit 2　ページ2（1のとき表示）
bit 3　ページ3（1のとき表示）

返り値

D0.l　ステータス

0　正常終了
－1　グラフィック画面が使えない
－2　ページ・ナンバーがおかしい
－3　指定されたページ・ナンバーは現在の画面モードでは使えない

機能

グラフィック画面の表示ページを設定します．引数のページ・ナンバーは絶対的なナンバーではなく，表示優先順位の高いものから順に付けたナンバーです（いちばん優先順位が高いページが0となる）．

コール

```
moveq   #$B2,D0
move.b  #%1011,D1
trap    #15
```

例

```
moveq   #$B2,D0
move.b  #%1100,D1
trap    #15
```

この場合，ページ2とページ3を表示する．

$B3 _HOME

引数

D1.b　ページ指定データ

bit 0　ページ0（1のとき座標を変更する）
bit 1　ページ1（1のとき座標を変更する）
bit 2　ページ2（1のとき座標を変更する）
bit 3　ページ3（1のとき座標を変更する）
0　すべてページを変更する

D2.w　X座標

D3.w　Y座標

返り値

D0.l　ステータス

0　正常終了
－1　グラフィック画面が使えない
－2　ページナンバーがおかしい

−3　　指定されたページナンバーは現在の画面モードでは使えない

機能

グラフィック画面の表示位置を変更します.

コール

```
moveq   #$B3,D0
move.b  #%1000,D1
move.w  #$100,D2
move.w  #$10,D3
trap    #15
```

サンプル・プログラム

P.460　IOCSサンプル24

$B4　_WINDOW

引数

D1.w　ウィンドウ左上のX座標
D2.w　ウィンドウ左上のY座標
D3.w　ウィンドウ右下のX座標
D4.w　ウィンドウ右下のY座標

返り値

D0.l　ステータス

0　　正常終了
−1　　グラフィック画面が使えない
−2　　座標指定がおかしい

機能

グラフィック画面のウィンドウを設定します. IOCSコールの$B0番台のものに有効です.

コール

```
moveq   #$B4,D0
move.w  #$12,D1
move.w  #$34,D2
move.w  #$56,D3
move.w  #$78,D4
trap    #15
```

$B5 _WIPE

引数

なし

返り値

D0.l　　−1のときエラー

機能

グラフィック画面をクリアします.

コール

```
        moveq   #$B5,D0
        trap    #15
```

$B6 _PSET

引数

A1.l　　パラメータデータ・アドレス
　　　　(A1).w　X座標
　　　　2(A1).w Y座標
　　　　4(A1).w パレットコード

返り値

D0.l　　−1のときエラー

機能

グラフィック画面に点を描きます.

コール

```
        moveq   #$B6,D0
        lea     datadd,A1
        trap    #15
```

$B7 _POINT

引数

A1.l　　パラメータデータ・アドレス
　　　　(A1).w　X座標
　　　　2(A1).w Y座標
　　　　4(A1).w 返り値のパレットコード用のワーク

返り値

4(A1).w にパレットコードがセットされる

D0.l　　−1のときエラー

機能

グラフィック画面の指定された座標のパレットコードを調べます.

コール

```
        moveq   #$B7,D0
        lea     datadd,A1
        trap    #15
```

$B8　_LINE

引数

A1.l　パラメータデータ・アドレス

(A1).w	始点X座標
2(A1).w	始点Y座標
4(A1).w	終点X座標
6(A1).w	終点Y座標
8(A1).w	パレットコード
$A(A1).w	ラインスタイル

返り値

D0.l　－1のときエラー

機能

グラフィック画面にラインを描きます.

コール

```
moveq   #$B8,D0
lea     datadd,A1
trap    #15
```

サンプル・プログラム

P.460　IOCSサンプル24

$B9　_BOX

引数

A1.l　パラメータデータ・アドレス

(A1).w	始点X座標
2(A1).w	始点Y座標
4(A1).w	終点X座標
6(A1).w	終点Y座標
8(A1).w	パレットコード
$A(A1).w	ラインスタイル

返り値

D0.l　－1のときエラー

機能

グラフィック画面にボックスを描きます.

コール

```
moveq   #$B9,D0
lea     datadd,A1
trap    #15
```

サンプル・プログラム

P.460　IOCSサンプル24

$BA　_FILL

引数

A1.l　パラメータデータ・アドレス
　(A1).w　始点X座標
　2(A1).w　始点Y座標
　4(A1).w　終点X座標
　6(A1).w　終点Y座標
　8(A1).w　パレットコード

返り値

D0.l　－1のときエラー

機能

グラフィック画面に塗り潰しボックスを描きます.

コール

```
        moveq   #$BA,D0
        lea     datadd,A1
        trap    #15
```

サンプル・プログラム

P.460　IOCSサンプル24

$BB　_CIRCLE

引数

A1.l　パラメータデータ・アドレス
　(A1).w　中心X座標
　2(A1).w　中心Y座標
　4(A1).w　半径
　6(A1).w　パレットコード
　8(A1).w　円弧開始角度(度)負の値を指定すると扇型を描く
　$A(A1).w　円弧終了角度(度)負の値を指定すると扇型を描く
　$C(A1).w　比率

返り値

D0.l　－1のときエラー

機能

グラフィック画面に円（及び楕円）を描きます.

コール

```
        moveq   #$BB,D0
        lea     datadd,A1
        trap    #15
```

サンプル・プログラム

P.460　IOCSサンプル24

$BC　_PAINT

引数

A1.l　パラメータデータ・アドレス

(A1).w	始点X座標
2(A1).w	始点Y座標
4(A1).w	パレットコード
6(A1).l	作業領域先頭アドレス
$A(A1).l	作業領域終了アドレス

返り値

D0.l　－1のときエラー

機能

グラフィック画面の指定座標をふくむ範囲を塗り潰します．作業領域は偶数アドレスから始まるようにしてください．作業領域不足の場合はペイント途中で戻ってきます．

コール

```
moveq   #$BC,D0
lea     datadd,A1
trap    #15
```

サンプル・プログラム

P.460　IOCSサンプル24

謎の"MEMDRV"

Human68K Ver2.00以降に付属するprocess.xを解析してみると，謎のメモリ・ブロックIDをみつけることができます．その名は"MEMDRV"．

process.xでは，ユーザー・モードで動いているプロセスは"USER"，スーパーバイザ・モードで動いているプロセスは"SUPER",常駐しているプロセスは"KEEP"と表示され，$FF48 mallocなどで確保されたメモリ・ブロックは"MALLOC"と表示されることになっています．それでは，"MEMDRV"は？

"MEMDRV"と表示されるべきメモリ・ブロックは，メモリ管理ポインタの「このメモリを確保したプロセスのメモリ管理ポンイタ」が$FExxxxxxという値であることになっているようです(この最上位8ビットは68000では意味を持たないので，メモリ・ブロックのIDとして使われています)．ところが，いくら捜しても，現時点ではこういうメモリ・ブロックのIDを設定するようなファンクション・コールは存在しません．

シャープさんは，いったい何を用意してくれているのでしょう？　楽しみですね．

$BD _SYMBOL

引数

A1.l　パラメータデータ・アドレス

(A1).w	X座標
2(A1).w	Y座標
4(A1).l	文字列データ先頭アドレス
8(A1).b	横方向の倍率
9(A1).b	縦方向の倍率
$A(A1).w	パレットコード
$C(A1).b	文字フォントのタイプ 0　全角が12×12ドット 1　全角が16×16ドット 2　全角が24×24ドット
$D(A1).b	回転データ 0　回転しない 1　90度回転 2　180度回転 3　270度回転

返り値

D0.l　－1のときエラー

機能

グラフィック画面に文字を表示します.

コール

```
	moveq	#$BD,D0
	lea	datadd,A1
	trap	#15
```

サンプル・プログラム

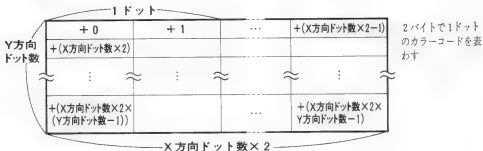
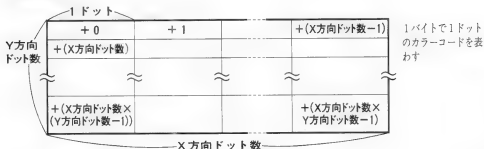
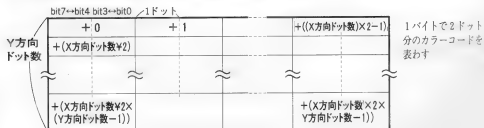
P.460　IOCSサンプル24

$BE　_GETGRM

引数

A1.1　パラメータデータ・アドレス

(A1).w	始点X座標
2(A1).w	始点Y座標
4(A1).w	終点X座標
6(A1).w	終点Y座標
8(A1).l	バッファ先頭アドレス（偶数アドレス）
$C(A1).l	バッファ終了アドレス

グラフィックVRAM読み書きデーターフォーマット

65536色モード

256色モード

16色モード（X方向ドット数が偶数の場合）

16色モード（X方向のドット数が奇数の場合）

基本的に偶数のときと同じ．ただし，右端に余る4ビット（半バイト＝1ドット）に次のラインの左端のドットのカラーコードが対応し，以降同様に詰めていく．

(例)

8	1	0	0	A
D	B	4	3	2
1	0	9	8	9

このパターンは次のようになる．

+0	+1	+2	+3	+4	+5	+6	+7
$81	$00	$AD	$B4	$32	$10	$98	$90

↑最後の0が余り

返り値

D0.l　　−1のときエラー

機能

グラフィック画面からデータを読み込みます．

コール

```
moveq   #$BE,D0
lea     datadd,A1
trap    #15
```

bind.xの未公開オプション

Human68K Ver2.0xに付属する，オーバーレイXファイル作成支援ツール，"bind.x"にも未公開オプションが存在します．

●/V

いわゆる，「バーボーズモード」の指定です．作業の内容を逐一表示してくれます．

●/A <file>

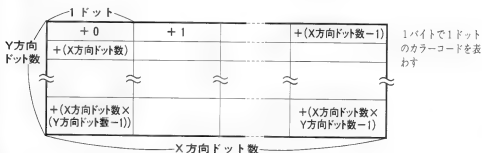
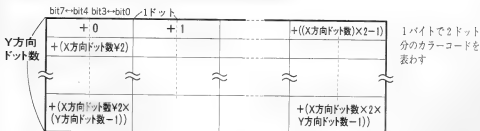
●/B

$BF _PUTGRM

引数

A1.1　パラメータデータ・アドレス

(A1).w	始点X座標
2(A1).w	始点Y座標
4(A1).w	終点X座標
6(A1).w	終点Y座標
8(A1).l	データ先頭アドレス（偶数アドレス）
$C(A1).l	データ終了アドレス

グラフィックVRAM読み書きデーターフォーマット

65536色モード

2バイトで1ドットのカラーコードを表わす

256色モード

1バイトで1ドットのカラーコードを表わす

16色モード（X方向ドット数が偶数の場合）

1バイトで2ドット分のカラーコードを表わす

16色モード（X方向のドット数が奇数の場合）

基本的に偶数のときと同じ．ただし，右端に余る4ビット（半バイト＝1ドット）に次のラインの左端のドットのカラーコードが対応し，以降同様に詰めていく．

(例)

8 1 0 0 A
D B 4 3 2
1 0 9 8 9

このパターンは次のようになる．

＋0　＋1　＋2　＋3　＋4　＋5　＋6　＋7

$81　$00　$AD　$B4　$32　$10　$98　$90

↑最後の0が余り

返り値

D0.l　　−1のときエラー

機能

グラフィック画面にデータを書き込みます．ウインドウにかかる位置に表示させようとすると，まったく表示されません．

コール

```
        moveq   #$BF,D0
        lea     datadd,A1
        trap    #15
```

$C0　　_SP_INIT

引数

なし

返り値

D0.l　　ステータス

　　0　　　正常終了

　−1　　　画面モードが正しくない

機能

スプライト画面を初期化します.

コール

```
        moveq   #$C0,D0
        trap    #15
```

サンプル・プログラム

P.461　IOCSサンプル25

$C1　　_SP_ON

引数

なし

返り値

D0.l　　ステータス

　　0　　　正常終了

　−1　　　画面モードが正しくない

機能

スプライト画面(スプライトとバックグラウンド)を表示します.

コール

```
        moveq   #$C1,D0
        trap    #15
```

サンプル・プログラム

IOCSサンプル24

$C2　　_SP_OFF

引数

なし

返り値

D0が破壊される

機能

スプライト画面(スプライトとバックグラウンド)を表示しないようにします.

コール

```
        moveq   #$C2,D0
        trap    #15
```

$C3　_SP_CGCLR

引数

D1.l　PCGコード（0～$FF)

返り値

D0.l　－1のときエラー

機能

指定したPCGをクリアします.

コール

```
        moveq   #$C3,D0
        move.l  #$D0,D1
        trap    #15
```

$C4　_SP_DEFCG

引数

D1.l　PCGコード（0～$FF)

D2.l　パターン・サイズ

　0　8×8ドットのパターン

　1　16×16ドットのパターン

A1.l　パターン・データ・アドレス(偶数アドレス)

　8×8ドットのとき32バイト

　16×16ドットの時128バイト

返り値

D0.l　－1のときエラー

機能

指定したPCGにパターンを定義します.

コール

```
        moveq   #$C4,D0
        move.l  #$40,D1
        moveq   #1,D2
        lea     dataad,A1
        trap    #15
```

サンプル・プログラム

P.461,P.463　IOCSサンプル25, 26

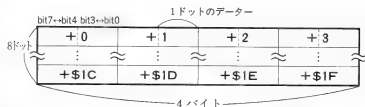
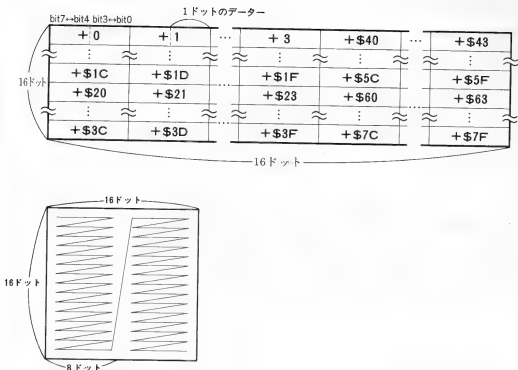
8×8ドット

16×16ドット

1ドットのデーター

bit7↔bit4 bit3↔bit0

+0	+1	…	+3	+$40	…	+$43
⋮	⋮		⋮	⋮		⋮
+$1C	+$1D	…	+$1F	+$5C		+$5F
+$20	+$21		+$23	+$60	…	+$63
⋮	⋮		⋮	⋮		⋮
+$3C	+$3D	…	+$3F	+$7C		+$7F

16ドット（縦） 16ドット（横）

バックグラウンド機能ファンクションのベクタ

プログラム実行中にファンクションコールの$FF??に出会った場合，1111系の未実装命令例外処理が発生し，Human内部のファンクションコール処理ルーチンに飛んできます．ここでは，$FFに続くファンクションコール番号に従って，$1800から存在するベクタを参照しベクタに設定されている各処理アドレスへと分岐します．

ファンクションコールはこのような方法で呼び出されているわけですが，$FFF8～$FFFFのバックグラウンド機能関係のファンクションコールは多少異なっています．

ファンクションコール処理ルーチンの中で，ファンクション番号が$F8～$FFであると判断された場合，ベクタは参照されず，それぞれのルーチンへ直接分岐してしまいます．そのため，ベクタを書き換えても分岐先を変更することはできなくなっています．まったくベクタが参照されないかというとそういうわけではなく，ファンクションコール処理ルーチンから直接分岐したところでの処理が終わったところで，初めて参照され，ベクタに従って分岐するようになっています．

これらのベクタを利用することによって，バックグラウンド処理を考慮した，ウインドウやマルチスクリーンを使った環境を構築することも可能に思えます．皆さんも考えてみてはいかがでしょうか．

$C5　_SP_GTPCG

引数

D1.l　PCGコード（0～$FF）

D2.l　パターン・サイズ

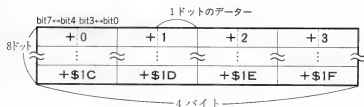
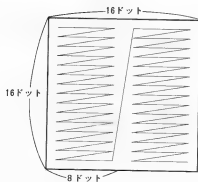
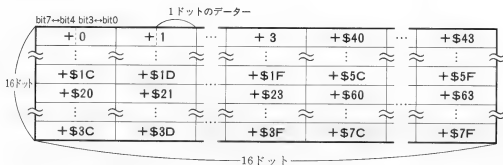
0　8×8ドットのパターン

1　16×16ドットのパターン

A1.l　パターン・データ・バッファアドレス（偶数アドレス）

8×8ドットのとき32バイト

16×16ドットのとき128バイト

8×8ドット

16×16ドット

返り値

D0.1　　－1のときエラー

機能

指定したPCGデータを読み込みます.

コール

```
        moveq   #$C5,D0
        move.l  #$25,D1
        moveq   #0,D2
        lea     bufadd,A1
        trap    #15
```

as.x のバグ

初代X68Kのシステム・ディスクに入っていた『福袋バージョン』から，最近のCコンパイラ/福袋2.0 に付属するバージョン1.01まで，as.xには共通のバグが潜んでいます.

次のようなプログラムをアセンブルしてみてください.

```
                text
lbl1:
                nop
lbl2:
                move.l  #lbl2-lbl1,D0
                dc.w    $ff00
```

実際に生成されたXファイルをdb.xで見てみると，次のようなコードが出力されていることがわかります.

```
lbl1:
  0008BCD0      nop
lbl2:
  0008BCD2      move.l  #$00000002,D0
  0008BCD8      ori.b   #$02,D0
  0008BCDC      _EXIT
```

点線部のように，まったく関係のないコードが出力されています.

このバグの症状を一般化すると，次のように表現できます.

イミディエイト・アドレッシングにおいて，ソースにラベル同士の計算式を，ディスティネーションにアドレス・レジスタ以外を指定した場合，異常なコードを出力することがある.

このバグを回避する方法は，すなわちこのようなプログラムを書かないことです．上の例であれば，move命令の行を次のように書き換えることでバグの発生を抑えることができます.

```
        lea     lbl2-lbl1,A0
        move.l  A0,D0
```

また，ラベルだけを独立行とせず，ラベルの後ろにニモニックを書くようにすると，このバグは発生しない模様です.

なお，このバグは少なくとも福袋バージョンとas.xバージョン1.01(タイムスタンプ88-04-02 12:00:00)で確認されています.

$C6　_SP_REGST

引数

D1.l　下位7ビット　スプライトナンバー（0～$7F)
　bit $1F　垂直帰線間の検出の有無（1のとき検出しない)

D2.l　X座標（0～$3FF）$16が実際の画面の左端に相当する
　−1のときは前回指定した値

D3.l　Y座標（0～$3FF）$16が実際の画面の上端に相当する
　−1のときは前回指定した値

D4.l　パターン・コード
　bit 0～7　PCGコード(0～$FF)
　bit 8～$B　カラーパレット・ブロックナンバー（0～$F)
　bit $E　横方向反転指定(1のとき反転)
　bit $F　縦方向反転指定(1のとき反転)

D5.l　プライオリティ
　0　スプライトを表示しない
　1　スプライトはBG0&BG1の後ろ
　2　スプライトはBG0とBG1の間
　3　スプライトはBG0&BG1より前
　−1　前回指定した値

返り値

D0.l　−1のときエラー

機能

スプライト・レジスタの設定を行ない，スプライトを表示します．指定したスプライトが，指定した座標で，指定されたパターン（PCG）を，指定したパレットを使って表示します．

コール

```
        moveq   #$C6,D0
        move.l  #$12,D1
        move.l  #$123,D2
        move.l  #$45,D3
        move.l  #$3*256+$51,D4
        move.l  #$3,D5
        trap    #15
```

サンプル・プログラム

P.231　IOCSサンプル25

$C7　_SP_REGGT

引数

D1.l　スプライトナンバー(0～$7F)

返り値

D0.l　－1のときエラー
D2.l　X座標(0～$3FF)　$16が実際の画面の左端に相当する
D3.l　Y座標(0～$3FF)　$16が実際の画面の上端に相当する
D4.l　パターン・コード
　bit 0～7　PCGコード（0～$FF)
　bit 8～$B　カラーパレット・ブロックナンバー（0～$F)
　bit $E　横方向反転指定（1のとき反転)
　bit $F　縦方向反転指定（1のとき反転)
D5.l　プライオリティ
　0　スプライトを表示しない
　1　スプライトはBG0&BG1の後ろ
　2　スプライトはBG0とBG1の間
　3　スプライトはBG0&BG1より前

機能

スプライト・レジスタの内容を読み出します.

コール

```
moveq   #$C7,D0
move.l  #$12,D1
trap    #15
```

$C8　_BGSCRLST

引数

D1.l　bit 0　設定するBGのナンバー（0,1)
　bit $1F　垂直帰線間の検出の有無（1のとき検出しない)
D2.l　X座標(0～$3FF)
　－1のときは前回指定した値
D3.l　Y座標(0～$3FF)
　－1のときは前回指定した値

返り値

D0.l　－1のときエラー

機能

BGスクロール・レジスタの設定を行ないます.

コール

```
moveq   #$C8,D0
move.l  #$1,D1
move.l  #$12,D2
move.l  #$34,D3
trap    #15
```

$C9　_BGSCRLGT

引数

D1.l　読み出すBGのナンバー (0,1)

返り値

D0.l　−1のときエラー

D2.l　X座標 (0～$3FF)

D3.l　Y座標 (0～$3FF)

機能

BGスクロール・レジスタの内容を読み出します.

コール

```
moveq   #$C9,D0
move.l  #$0,D1
trap    #15
```

$CA　_BGCTRLST

引数

D1.l　設定するBGのナンバー(0,1)

D2.l　テキスト・ページナンバー(0,1)

　　　−1のときは前回指定したナンバー

D3.l　0　非表示

　　　1　表示

　　　−1のときは前回指定した値

返り値

D0.l　−1のときエラー

機能

BGコントロール・レジスタの設定を行ない，BGを表示します．指定したBGが，指定したBGテキストを表示します．

コール

```
moveq   #$CA,D0
move.l  #$0,D1
move.l  #$1,D2
move.l  #$1,D3
trap    #15
```

サンプル・プログラム

P.463　IOCSサンプル26

$CB　_BGCTRLGT

引数

D1.l　読み出すBGのナンバー (0,1)

返り値

D0.l　bit 0　1のとき表示，0のとき非表示
　　　bit 1　テキスト・ページナンバー (0,1)
　　　−1のときエラー

機能

BGコントロール・レジスタの内容を読み出します．

コール

```
moveq   #$CB,D0
move.l  #$1,D1
trap    #15
```

$CC　_BGTEXTCL

引数

D1.l　クリアするBGテキスト・ページナンバー (0,1)

D2.l　パターン・コード
　　　bit 0〜7　PCGコード(0〜$FF)
　　　bit 8〜$B　カラーパレット・ブロックナンバー (0〜$F)
　　　bit $E　横方向反転指定(1のとき反転)
　　　bit $F　縦方向反転指定(1のとき反転)

返り値

D0.l　−1のときエラー

機能

BGテキストを指定したパターン・コードで埋めます．

コール

```
moveq   #$CC,D0
move.l  #$1,D1
move.l  #$0,D2
trap    #15
```

サンプル・プログラム

P.463　IOCSサンプル26

$CD　_BGTEXTST

引数

D1.l　設定するBGテキスト・ページナンバー (0,1)
D2.l　X座標 (0～$3F)
D3.l　Y座標 (0～$3F)
D4.l　パターン・コード
　bit 0～7　PCGコード(0～$FF)
　bit 8～$B　カラーパレット・ブロックナンバー(0～$F)
　bit $E　横方向反転指定 (1のとき反転)
　bit $F　縦方向反転指定 (1のとき反転)

返り値

D0.l　－1のときエラー

機能

BGテキストにデータを書き込みます．

コール

```
moveq   #$CD,D0
move.l  #$1,D1
move.l  #$20,D2
move.l  #$15,D3
move.l  #$6A,D4
trap    #15
```

$CE　_BGTEXTGT

引数

D1.l　読み出すBGテキスト・ページナンバー (0,1)
D2.l　X座標 (0～$3F)
D3.l　Y座標 (0～$3F)

返り値

D0.l　パターン・コード
　bit 0～7　PCGコード (0～$FF)
　bit 8～$B　カラーパレット・ブロックナンバー(0～$F)
　bit $E　横方向反転指定(1のとき反転)
　bit $F　縦方向反転指定(1のとき反転)
　－1のときエラー

機能

BGテキストからデータを読み出します．

コール

```
moveq   #$CE,D0
move.l  #$1,D1
move.l  #$12,D2
move.l  #$30,D3
trap    #15
```

$CF _SPALET

引数

D1.l　下位４ビット　パレットコード(０～$F)
　　　bit $1F　垂直帰線期間の検出の有無(１のとき検出しない)

D2.l　パレット・ブロックナンバー（１～$F)

D3.l　カラーコード（０～$FFFF)
　　　−1のときデータを読み出す

返り値

D0.l　設定前の（D3.l＝−1のときは現在の）カラーコード（０～$FFFF)
　　　−1のときエラー
　　　−2のときパレット・ブロック０を指定した

機能

スプライト・パレットの設定・読み出しを行ないます．

コール

```
        moveq   #$CF,D0
        move.l  #$2,D1
        move.l  #$4,D2
        move.l  #$1234,D3
        trap    #15
```

サンプル・プログラム

P.461　IOCSサンプル25

$D3 _TXXLINE

引数

A1.l　パラメータデータ・アドレス

(A1).w　テキスト・プレーンナンバー

0	$E00000～$E1FFFF
1	$E20000～$E3FFFF
2	$E40000～$E5FFFF
3	$E60000～$E7FFFF

2(A1).w　Xドット座標

4(A1).w　Yドット座標

6(A1).w　水平線の長さ

8(A1).w　ラインスタイル

返り値

D0が破壊される

機能

テキスト画面に水平線を描きます.

コール

```
moveq   #$D3,D0
lea     datadd,A1
trap    #15
```

$D4 _TXYLINE

引数

A1.l　パラメータデータ・アドレス

(A1).w　テキスト・プレーンナンバー

0	$E00000～$E1FFFF
1	$E20000～$E3FFFF
2	$E40000～$E5FFFF
3	$E60000～$E7FFFF

2(A1).w　Xドット座標

4(A1).w　Yドット座標

6(A1).w　垂直線の長さ

8(A1).w　ラインスタイル

返り値

D0が破壊される

機能

テキスト画面に垂直線を描きます.

コール

```
moveq   #$D4,D0
lea     datadd,A1
trap    #15
```

$D6 _TXBOX

引数

A1.1　パラメータデータ・アドレス

(A1).w	テキスト・プレーンナンバー	
	0	$E00000～$E1FFFF
	1	$E20000～$E3FFFF
	2	$E40000～$E5FFFF
	3	$E60000～$E7FFFF
2(A1).w	Xドット座標	
4(A1).w	Yドット座標	
6(A1).w	矩形のX方向長さ	
8(A1).w	矩形のY方向長さ	
$A(A1).w	ラインスタイル	

返り値

D0が破壊される

機能

テキスト画面にボックスを描きます.

コール

```
moveq   #$D6,D0
lea     datadd,A1
trap    #15
```

サンプル・プログラム

P.464　IOCSサンプル27

$D7　_TXFILL

引数

A1.l　パラメータデータ・アドレス

(A1).w	テキスト・プレーンナンバー	
	0	$E00000～$E1FFFF
	1	$E20000～$E3FFFF
	2	$E40000～$E5FFFF
	3	$E60000～$E7FFFF
2(A1).w	Xドット座標	
4(A1).w	Yドット座標	
6(A1).w	矩形のX方向長さ	
8(A1).w	矩形のY方向長さ	
$A(A1).w	ペイントスタイル	

返り値

D0が破壊される

機能

テキスト画面に塗り潰しボックスを描きます.

コール

```
moveq   #$D7,D0
lea     datadd,A1
trap    #15
```

サンプル・プログラム

P.464　IOCSサンプル27

$D8　_TXREV

引数

A1.l　パラメータデータ・アドレス

(A1).w	テキスト・プレーンナンバー	
	0	$E00000～$E1FFFF
	1	$E20000～$E3FFFF
	2	$E40000～$E5FFFF
	3	$E60000～$E7FFFF
2(A1).w	Xドット座標	
4(A1).w	Yドット座標	
6(A1).w	矩形のX方向長さ	
8(A1).w	矩形のY方向長さ	

返り値

D0が破壊される

機能

テキスト画面の矩形域内を反転します.

コール

```
        moveq   #$D8,D0
        lea     datadd,A1
        trap    #15
```

サンプル・プログラム

P.464　IOCSサンプル27

$DF　_TXRASCPY

引数

D1.w　上位8ビット　コピー元ラスタ・ナンバー（1ラスタ=4ライン）
　　　下位8ビット　コピー先ラスタ・ナンバー（1ラスタ=4ライン）
D2.w　コピーラスタ数
D3.w　上位8ビット　コピーラスタ・ポインタ移動方向
　　　　　　　　　0　　下方向
　　　　　　　　　$FF　上方向
　　　下位8ビット　コピーするプレーン指定データ
　　　　　　　　　bit 0　1のときプレーン0のコピーを行なう
　　　　　　　　　bit 1　1のときプレーン1のコピーを行なう
　　　　　　　　　bit 2　1のときプレーン2のコピーを行なう
　　　　　　　　　bit 3　1のときプレーン3のコピーを行なう

返り値

D0が破壊される

機能

テキスト画面の指定した部分をラスタ・コピーによってコピーします．X方向はすべてコピーされます．

コール

```
        moveq   #$DF,D0
        lea     datadd,A1
        trap    #15
```

サンプル・プログラム

P.464　IOCSサンプル27

$FD　_ABORTRST

引数

なし

返り値

D0 が破壊される

機能

アボートするための環境を再設定します．通常は使いません．

コール

```
        moveq   #$FD,D0
        trap    #$15
```

$FE　_IPLERR

引数

なし

返り値

なし

機能

起動時のエラーで再起動するときに使います．通常は使いません．

コール

```
moveq   #$FE,D0
trap    #15
```

$FF　_ABORTJOB

引数

なし

返り値

なし

機能

アボートします．通常は使いません．

コール

```
moveq   #$FF,D0
trap    #$15
```

4章 ROM以外のIOCSコール

IOCS コールのルーチンは ROM に書かれていますが，IOCS $80 によって新規定義，変更ができます．

4.1 ROM 以外の IOCS コールの概要

IOCSコールはROMに処理ルーチンがありますが，ベクタ・テーブル（処理先アドレス表）がRAM上にあります．このため，ベクタ・テーブルを書き換えて，処理ルーチンをRAM上におくことができます．

IOCSコールではIOCS $80がベクタ書き換えの機能を持っています．よってIOCS $80を使うことによって，新しいIOCSコールの登録，IOCSコールの変更などが可能です．

実際の例を挙げると，HUMAN.SYSによるシステム・アボート関係の変更，PRNDRV.SYSによるプリンタ関係の変更，OPMDRV.XによるOPM演奏機能の追加，AJOY.Xによるアナログジョイスティック操作機能の追加などがあります．

4.2 OPMDRV.X による IOCS コール

OPMDRV.XはOPMのファイル名でFM音源を演奏させるデバイス・ドライバですが，IOCSコールに演奏用サブルーチンを付け加えています．このIOCSコールはX-BASICやX-Cのオブジェクトなどによって利用されています．

OPMDRV.Xは$F0のIOCSコールを追加します．そして，OPMDRV.X内部にジャンプしてきた後，D1レジスタによってそれぞれの処理アドレスへ分岐するようになっています．

各IOCSコールの処理内容はX-BASICやX-Cのそれと同じです．また，演奏手順も同じとなっています．

$F0 $00 _M_INIT

引数

D1.l　0

返り値

なし

機能

ドライバの初期化を行ないます．

コール

```
moveq   #$F0,D0
moveq   #0,D1
trap    #15
```

$F0　$01　_M_ALLOC

引数

D1.l　1

D2.l　上位16ビット　トラック番号（1～$50）

　　　下位16ビット　（トラック・サイズ）－1（0～$FFFF)

返り値

D0.l　－1のときエラー

機能

トラック・バッファを確保します．トラックのデータはクリアされます．

コール

```
        moveq   #$F0,D0
        moveq   #1,D1
        move.l  #1*65536+5000,D2
        trap    #15
```

$F0　$02　_M_ASSIGN

引数

D1.l　2

D2.l　上位16ビット　チャンネル番号（1～8）

　　　下位16ビット　トラック番号（1～$50）

返り値

D0.l　－1のときエラー

機能

FM音源のチャンネルにトラックを割り当てます．

コール

```
        moveq   #$F0,D0
        moveq   #2,D1
        move.l  #1*65536+1,D2
        trap    #15
```

$F0　$03　_M_VGET

引数

D1.l　3
D2.l　音色番号（1～200)
A1.l　バッファアドレス

返り値

D0.l　－1のときエラー

OPMDRV.X の音色データフォーマット

オフセット		
+0	bit 3～5＝フィードバック　bit 0～2＝アルゴリズム	
+1	スロットON/OFF　bit 0＝M1　bit 1＝C1　bit 2＝M2　bit 3＝C2	
+2	ウェーブフォーム	
+3	シンクロON/OFF	
+4	スピード	
+5	PMD	
+6	AMD	
+7	PMS	
+8	AMS	
+9	左右ON/OFF	
+$A	未使用	
+$B	AR	M1
+$C	D1R	M1
+$D	D2R	M1
+$E	RR	M1
+$10	D1L	M1
+$11	TL	M1
+$12	KS	M1
+$13	MUL	M1
+$14	DT1	M1
+$15	DT2	M1
+$16	AMSフラグ	M1
+$17～+$22	M1と同形式	C1
+$23～+$2F	M1と同形式	M2
+$30～+$3B	M1と同形式	C2

機能

指定した音色データを読み込みます.

コール

```
moveq   #$F0,D0
moveq   #3,D1
moveq   #5,D2
lea     bufadd,A1
trap    #15
```

$F0　$04　_M_VSET

引数

D1.l　　4

D2.l　　音色番号（1～200）

A1.l　　データ・アドレス

返り値

D0.l　　－1のときエラー

機能

音色を定義します.

コール

```
moveq   #$F0,D0
moveq   #4,D1
moveq   #5,D2
lea     datadd,A1
trap    #15
```

$F0　$05　_M_TEMPO

引数

D1.l　　5

D2.l　　テンポ（32～200）

返り値

D0.l　　－1のときエラー

機能

テンポを設定します.

コール

```
moveq   #$F0,D0
moveq   #5,D1
move.l  #120,D2
trap    #15
```

$F0　$06　_M_TRK

引数

D1.l　6

D2.l　トラック番号（1～$50）

A1.l　データ・アドレス（X-BASICのMMLと同一，最後に$00をつける）

返り値

D0.l　－1のときエラー

機能

指定したトラックにMMLデータをセットします．

コール

```
        moveq   #$F0,D0
        moveq   #6,D1
        moveq   #1,D2
        lea     mmldat,A1
        trap    #15
```

$F0　$07　_M_FREE

引数

D1.l　7

D2.l　トラック番号（1～$50）

返り値

D0.l　トラック残りバイト数

－1のときエラー

機能

指定したトラックの空き容量を調べます．

コール

```
        moveq   #$F0,D0
        moveq   #7,D1
        moveq   #1,D2
        trap    #15
```

$F0 $08 _M_PLAY

引数

D1.l　8

D2.l　bit 0　チャンネル1

bit 1　チャンネル2

bit 2　チャンネル3

bit 3　チャンネル4

bit 4　チャンネル5

bit 5　チャンネル6

bit 6　チャンネル7

bit 7　チャンネル8

1のとき，そのチャンネルの演奏開始

すべて0なら全チャンネル演奏開始

返り値

D0.l　−1のときエラー

機能

指定したチャンネルの演奏を開始します．

コール

```
moveq   #$F0,D0
moveq   #8,D1
moveq   #0,D2
trap    #15
```

$F0 $09 _M_STAT

引数

D1.l　9

D2.l　チャンネル番号（1～8）

0のときは全チャンネルを調べる

返り値

D0.l　0のとき指定されたチャンネルは演奏していない（D2.l <> 0）

bit 0　チャンネル1

bit 1　チャンネル2

bit 2　チャンネル3

bit 3　チャンネル4

bit 4　チャンネル5

bit 5　チャンネル6

bit 6　チャンネル7

bit 7　チャンネル8

1のときそのチャンネルは演奏中（D2.l=0）

−1のならエラー

機能

チャンネルの演奏状態を調べます.

コール

```
moveq   #$F0,D0
moveq   #9,D1
moveq   #3,D2
trap    #15
```

$FO　$OA　_M_STOP

引数

D1.l　$A

D2.l　bit 0　チャンネル1
　　　bit 1　チャンネル2
　　　bit 2　チャンネル3
　　　bit 3　チャンネル4
　　　bit 4　チャンネル5
　　　bit 5　チャンネル6
　　　bit 6　チャンネル7
　　　bit 7　チャンネル8
　　　1なら，そのチャンネルの演奏中止
　　　すべて0のときは全チャンネル演奏中止

返り値

D0.l　－1のときエラー

機能

指定したチャンネルの演奏を中止します.

コール

```
moveq   #$F0,D0
moveq   #$A,D1
moveq   #0,D2
trap    #15
```

$F0　$0B　_M_CONT

引数

D1.l　　$B

D2.l　　bit 0　チャンネル1
　　　　bit 1　チャンネル2
　　　　bit 2　チャンネル3
　　　　bit 3　チャンネル4
　　　　bit 4　チャンネル5
　　　　bit 5　チャンネル6
　　　　bit 6　チャンネル7
　　　　bit 7　チャンネル8
　　　　1のとき，そのチャンネルの演奏再開
　　　　すべて0のときは全チャンネル演奏再開

返り値

D0.l　　−1のときエラー

機能

指定したチャンネルの演奏を再開します．

コール

```
        moveq   #$F0,D0
        moveq   #$B,D1
        move.l  #%00011011,D2
        trap    #15
```

$F0　$0C　_M_ATOI

引数

D1.l　　$C

D2.l　　チャンネル番号（1～8）

返り値

D0.l　　トラック・バッファアドレス

機能

指定したチャンネルに対応したトラック・バッファのアドレスを調べます．

コール

```
        moveq   #$F0,D0
        moveq   #$C,D1
        moveq   #
```

OPMDRV.XのIOCSコールのサンプル・プログラムはP.464 IOCSサンプル28

4.3 AJOY.XによるIOCSコール

AJOY.Xはアナログジョイスティック用IOCSを組み込むデバイス・ドライバです．このデバイス・ドライバは$F2のIOCSコールを追加します．そして，AJOY.X内部にジャンプしてきた後，D1レジスタによってそれぞれの処理アドレスへ分岐するようになっています．

$F2　$00

引数

D1.l　　0

A1.l　　バッファアドレス

アナログ・ジョイステックのデータ・フォーマット

オフセット	データ
+0.w	ステック上下(上　0～$FF　下)
+2.w	ステック左右(左　0～$FF　右)
+4.w	スロットル
+6.w	オプション
+8.w	トリガを押しているとき0になる
	bit 0　セレクト
	bit 1　スタート
	bit 2　E2
	bit 3　E1
	bit 4　D
	bit 5　C
	bit 6　B, B'(どちらかを押していると0)
	bit 7　A, A'(どちらかを押していると0)
	bit 8　B'
	bit 9　A'
	bit $A　B
	bit $B　A

返り値

D0.l　　−1のときエラー

正常に受信できたとき，バッファにデータが書き込まれる

機能

アナログジョイスティック・データを読み込みます．

コール

```
moveq   #$F2,D0
moveq   #0,D1
lea     bufadd,A1
trap    #15
```

$F2　$01

引数

D1.l　　1

D2.w　　0　　　デジタル・モード

　　　　1　　　アナログ・モード

　　　　－1　　現在のモードを調べる

返り値

D0.l　　現在，または設定前のモード

機能

ジョイスティック・モードを調べます．

コール

```
        moveq   #$F2,D0
        moveq   #1,D1
        move.w  #1,D2
        trap    #15
```

$F2　$02

引数

D1.l　　2

D2.w　　通信速度

　　　　0　　　最高速度

　　　　1　　　最高速度の1/2

　　　　2　　　最高速度の1/3

　　　　3　　　最高速度の1/4

　　　　－1　　現在の速度を調べる

返り値

D0.l　　現在，または設定前の速度

機能

ジョイスティックの通信速度を設定します．最高速度に設定した場合，ジョイスティックをリセットしない限り他の速度に変更できません．

コール

```
        moveq   #$F2,D0
        moveq   #2,D0
        clr.w   D2
        trap    #15
```

AJOY.XのIOCSコールのサンプル・プログラムはP.465 IOCSサンプル29

＊．R タイプの実行ファイルの作成方法

Human68Kの実行型ファイルには次の3種類があります．

＊．X 最もよく使われている型でソフト・リロケータブル・ファイルです．ファイルの最後にリロケート情報がついており，システムがリロケート作業を行なった後に実行されます．

＊．R プログラムの側でリロケータブルに作成された型で，プログラム以外のデータはついていません．システムはロードした後，何もせずに実行します．

＊．Z 特定のアドレスにロードされる型で，ロードするアドレスなどの情報がついています．通常は使いません．

(このうち，＊．X型ファイルにはシンボル・データつきのものとついていないものがあります)．

これらのファイルのなかで，＊．X，＊．Z型のファイルは簡単に作成できますが，＊．R型のファイルは作成時にリロケータブルにする必要があります．リロケータブルにするにはさまざまな方法がありますが，簡単な例を紹介します．

まず，アドレス・レジスタを1つワーク用に固定して，それ以外の用途に使わないようにします．通常，A6レジスタにします．そして，プログラム先頭でワークエリアの先頭アドレスを代入し，それ以後は値を変化させないようにします．そして，ワークエリアへのアクセスはA6レジスタ相対アドレッシングを使います．また，ジャンプ，サブルーチン・コールは相対の命令を使います．

この考え方でプログラムを作成すれば＊．R型にコンバートできるプログラムになりますが，プログラム・サイズやワーク・サイズが大きくなると相対アドレッシングが届かなくなってしまいます．この場合は，ワーク・レジスタを増やしたり，大きなワークのアクセスはその都度アドレス・レジスタを設定したり，無理矢理相対ジャンプさせたりする方法を取るしかありません．

〈例〉

```
            プログラム先頭
            lea      work(PC),A6             A6レジスタにワークエリアの値を代入
            move.W   work1(A6),D0            A6相対でアクセスする
            move.l   D7,work0(A6)
            lea      largework(PC), A5       大きなワークはアクセスするたびにアドレッシング作業を行
                                             なう．
            lea      pcadd(PC),A0            無理矢理相対ジャンプ
            move.l   #jumpadd-pcadd,D0
            adda.l   D0,A0
pcadd
            jmp      (A0)
jmpadd
            .bss
work
            ds.b     16384
largework
            ds.b     256*256
            .offset  0
work0
            ds.l     1
work1
            ds.w     1
            .end
```

IOCSサンプル1

```
*
*
*       ＩＯＣＳ＄００の返り値表示プログラム
*
*

*       使用方法
*               実行すると、ＩＯＣＳ＄００コールの返り値を表示します。
*               「￥」キーで終了します。

nextkey
                moveq   #0,D0                   * キー入力
                trap    #15

                cmpi.b  #'￥',D0                * 「￥」なら終了
                beq     return

                moveq   #4-1,D3                 * D0.w の内容を表示
                move.w  D0,D2
loop
                rol.w   #4,D2
                move.w  D2,D0
                andi.w  #$F,D0
                addi.b  #'0',D0
                cmpi.b  #'9'+1,D0
                bcs     decimal
                addq.b  #7,D0
decimal
                move.w  D0,D1
                moveq   #$20,D0
                trap    #15
                dbra    D3,loop

                lea     space(PC),A1            * 間隔をとる
                moveq   #$21,D0
                trap    #15

                bra     nextkey                 * 再び入力待ち
return
                lea     crlf(PC),A1             * 改行する
                moveq   #$21,D0
                trap    #15

                dc.w    $FF00

space
                dc.b    '    ',0
crlf
                dc.b    13,10,0

                .end
```

IOCSサンプル2

```
*
*
*       ソフト的キー入力発生
*
*

*       使用方法
*               実行すると、自動的にキー入力が発生します。

                lea     keylist(PC),A0

loop
                move.b  (A0)+,D1
                beq     return

                moveq   #5,D0                   * キー入力発生
                trap    #15
                bra     loop
```

```
return
                dc.w    $FF00

keylist
                dc.b    $20                     * D
                dc.b    $18                     * I
                dc.b    $70                     * SHIFT
                dc.b    $14                     * R
                dc.b    $F0                     * SHIFT 離す
                dc.b    $1D                     * RETURN
                dc.b    0

                .end
```

IOCSサンプル3

```
*
*
*       キーリピート関係の速度設定
*
*

*       使用方法
*               <キーリピートまでの時間> <キーリピート間隔>
*           それぞれ0からFまでの16進数

                tst.b   (A2)+
                beq     return

                move.b  (A2)+,D1                * 最初の文字を取り出す
                subi.b  #'0',D1                 * 数値に変換する
                cmpi.b  #$A,D1                  *       D1.b に数値がセットされる
                bcs     decimal
                subq.b  #7,D1
                cmpi.b  #$10,D1
                bcs     decimal
                subi.b  #$20,D1
decimal
                moveq   #8,D0                   * キーリピートまでの時間を設定
                trap    #15

space
                move.b  (A2)+,D1                * 次の文字を取り出す
                beq     return
                cmpi.b  #' ',D1
                beq     space
                subi.b  #'0',D1                 * 数値に変換する
                cmpi.b  #$A,D1                  *       D1.b に数値がセットされる
                bcs     decimal2
                subq.b  #7,D1
                cmpi.b  #$10,D1
                bcs     decimal2
                subi.b  #$20,D1
decimal2
                moveq   #9,D0                   * キーリピート間隔を設定
                trap    #15

return
                dc.w    $FF00

                .end
```

IOCSサンプル4

```
*
*
*       専用CRTコントロール
*
*

*       使用方法
*               <コントロールコマンド>
*           コントロールコマンドは0から$3Fまでの16進数

                tst.b   (A2)+
                beq     return
```

```
        move.b  (A2)+,D0                * 数値に変換する
        subi.b  #'0',D0                 *       D1.b に数値がセットされる
        cmpi.b  #$A,D1
        bcs     decimal
        subq.b  #7,D1
        cmpi.b  #$10,D1
        bcs     decimal
        subi.b  #$20,D1
decimal
        move.b  (A2)+,D1
        beq     decimal2
        lsl.b   #4,D0
        subi.b  #'0',D1
        cmpi.b  #$A,D1
        bcs     decimal2
        subq.b  #7,D1
        cmpi.b  #$10,D1
        bcs     decimal2
        subi.b  #$20,D1
decimal2
        add.b   D0,D1

        moveq   #$C,D0                  * コントロールする
        trap    #15

return
        dc.w    $FF00

        .end
```

IOCSサンプル5

```
*
*
*       拡大画面
*
*

*       使用方法
*               実行すると、256×256ドットモードになります。
*               SCREEN で元に戻ります。

                moveq   #$10,D0                 * 画面モード切り替え
                moveq   #2,D1
                trap    #15

                moveq   #$2E,D0                 * スクロール範囲指定
                moveq   #0,D1
                move.l  #31*65536+15,D2
                trap    #15

                dc.w    $FF00

                .end
```

IOCSサンプル6

```
*
*
*       ソフトウェアキーボード&電卓の色を変化させる
*
*

*       使用方法
*               実行すると、ソフトウェアキーボード&電卓の色が、
*               リアルタイムに変化します。
*               もう一度実行すると終了します。

startadd
                bra     prog

check
                dc.b    '判別データー'

prog
                movea.l (A0),A1                 * 新しく呼ばれたか調べる
```

```
                adda.l  #$100+2,A1
                lea     check(PC),A2
                moveq   #12-1,D0
loop
                cmpm.b  (A1)+,(A2)+
                bne     start
                dbra    D0,loop

                                                * 終了処理
                clr.l   A1                      * 割り込み終了
                moveq   #$6B,D0
                trap    #15

                move.l  (A0),D0                 * 常駐していたメモリを開放
                addi.l  #$10,D0
                move.l  D0,-(SP)
                dc.w    $FF49
                addq.l  #4,SP

                dc.w    $FF00
start
                moveq   #$6B,D0                 * 割り込み開始
                move.w  #7*256+200,D1
                lea     intadd(PC),A1
                trap    #15

                clr.b   hsv_h

                clr.w   -(SP)                   * 常駐する
                move.l  #endadd-startadd+1,-(SP)
                dc.w    $FF31
intadd
                movem.l D0-D2,-(SP)             * レジスタ保存
                move.b  hsv_h(PC),D1            * ＨＳＶデーター作成
                addq.b  #1,D1
                cmpi.b  #$C0,D1
                bcs     line0
                clr.b   D1
line0
                move.b  D1,hsv_h
                lsl.l   #8,D1
                lsl.l   #8,D1
                move.w  #$1F1F,D1
                moveq   #$12,D0                 * ＲＧＢコードに変換する
                trap    #15
                move.w  D0,D2
                ext.l   D2
                moveq   #4,D1                   * ソフトウェアキーボード&電卓の
                moveq   #$13,D0                 * ベースの色を変える
                trap    #15
                movem.l (SP)+,D0-D2
                rte
hsv_h
                dc.b    1
endadd
                .end
```

IOCSサンプル7

```
*
*
*       テキストパレットを異なった色にする
*
*

*       使用方法
*               実行すると、テキストパレットが異なった色になります。
*               SCREEN で元に戻ります。

                moveq   #0,D1
                lea     color(PC),A1
loop
                move.w  (A1)+,D2
                ext.l   D2
```

```
        moveq   #$14,D0                 * テキストパレット定義
        trap    #15

        addq.b  #1,D1
        cmpi.b  #$10,D1
        bcs     loop

        dc.w    $FF00

color
        dc.w    %00000_00000_00000_0
        dc.w    %00000_00000_11111_0
        dc.w    %00000_11111_00000_0
        dc.w    %00000_11111_11111_0
        dc.w    %11111_00000_00000_0
        dc.w    %11111_00000_11111_0
        dc.w    %11111_11111_00000_0
        dc.w    %11111_11111_11111_0

        dc.w    %11111_11111_11111_0
        dc.w    %11111_11111_00111_0
        dc.w    %11111_00111_11111_0
        dc.w    %11111_00111_00111_0
        dc.w    %00111_11111_11111_0
        dc.w    %00111_11111_00111_0
        dc.w    %00111_00111_11111_0
        dc.w    %00111_00111_00111_0

        .end
```

IOCSサンプル8

```
*
*
*       フォントデーターの種類表示
*
*

*       使用方法
*               実行すると、ＩＯＣＳレベルで使えるフォントをすべて表示します。

        moveq   #$20,D0                 * 画面クリア
        move.w  #$1A,D1
        trap    #15

        lea     $E20000,A0

        moveq   #6,D4
        bsr     put
        moveq   #8,D4
        bsr     put
        moveq   #$C,D4
        bsr     put

        moveq   #$23,D0                 * カーソルを下の方に移動
        move.w  #0,D1
        move.w  #25,D2
        trap    #15

        dc.w    $FF00

put
        move.w  #'X',D1                 * フォントアドレスを読み出す
        move.l  D4,D2
        moveq   #$16,D0
        trap    #15

        bsr     write                   * テキストＶＲＡＭに書き込む

        move.w  #'電',D1                * フォントアドレスを読み出す
        move.l  D4,D2
        moveq   #$16,D0
        trap    #15

        bsr     write                   * テキストＶＲＡＭに書き込む

        rts
```

```
write
                movea.l D0,A1
                movea.l A0,A2

                move.l  #128,D3
                ext.l   D1
                sub.l   D1,D3
                subq.l  #1,D3
                moveq   #$18,D0
                trap    #15

                adda.l  #4096,A0
                rts

                .end
```

IOCSサンプル9

```
*
*
*       24ドット文字のドット単位移動
*
*

*       使用方法
*               実行すると、「電」の文字が画面左上から右下へ移動します。
*               移動は7ドット単位です。クリッピング処理が行われます。

                move.l  #$C*65536+'電',D1      * フォントデーター読み込み
                lea     buffer(PC),A1
                moveq   #$19,D0
                trap    #15

                moveq   #$15,D0                * 書き込みプレーン指定
                moveq   #1,D1
                trap    #15

                moveq   #0,D1
                moveq   #0,D2
                lea     buffer(PC),A1
                lea     clipdata(PC),A2
loop
                moveq   #$1C,D0                * 書き込み
                trap    #15

                addq.w  #7,D1                  * 移動
                addq.w  #7,D2

                cmpi.w  #512,D2
                bcs     loop

                dc.w    $FF00
clipdata
                dc.w    100,100,450,450
buffer
                ds.b    2+2+3*24

                .end
```

IOCSサンプル10

```
*
*
*       テキスト画面部分コピー
*
*

*       使用方法
*               実行すると、テキスト画面の左下付近が右上にリアルタイムでコピーされます。
*               もう一度実行すると終了します。

startadd
                bra     prog
```

```
check
        dc.b    '判定データー'

prog
        movea.l (A0),A1                 * 新しく呼ばれたか調べる
        adda.l  #$100+2,A1
        lea     check(PC),A2
        moveq   #12-1,D0
loop
        cmpm.b  (A1)+,(A2)+
        bne     start
        dbra    D0,loop

                                        * 終了処理
        clr.l   A1                      * 割り込み終了
        moveq   #$6C,D0
        trap    #15

        move.l  (A0),D0                 * 常駐していたメモリを開放
        addi.l  #$10,D0
        move.l  D0,-(SP)
        dc.w    $FF49
        addq.l  #4,SP

        dc.w    $FF00

start
        moveq   #$6C,D0                 * 割り込み開始
        move.w  #0*256+5,D1
        lea     intadd(PC),A1
        trap    #15

        clr.w   -(SP)                   * 常駐する
        move.l  #endadd-startadd+1,-(SP)
        dc.w    $FF31

intadd
        movem.l D0-D2/A1,-(SP)          * レジスタ保存
        moveq   #2,D1                   * 読み書きするプレーンを設定
        moveq   #$15,D0
        trap    #15
        bsr     copy
        moveq   #1,D1                   * 読み書きするプレーンを設定
        moveq   #$15,D0
        trap    #15
        bsr     copy
        movem.l (SP)+,D0-D2/A1
        rte

copy
        moveq   #$1A,D0                 * コピー
        move.w  #3,D1
        move.w  #241,D2
        lea     buffer(PC),A1
        trap    #15
        moveq   #$1B,D0
        move.w  #585,D1
        move.w  #3,D2
        lea     buffer(PC),A1
        trap    #15
        rts

buffer
        dc.w    131
        dc.w    38
        ds.b    646
endadd
        .end
```

IOCSサンプル11

```
*
*
*       テキスト画面表示位置移動
*
*

*       使用方法
*               実行すると、テキスト画面の表示位置が３ドットずれます。
*               SCREEN で元に戻ります。

                moveq   #$1D,D0                 * ずらす
                moveq   #8,D1
                move.w  #3,D2
                move.w  #3,D3
                trap    #15
                dc.w    $FF00

                .end
```

IOCSサンプル12

```
*
*
*       表示範囲の変更
*
*

*       使用方法
*               実行すると画面中央部だけか表示範囲となります。
*               SCREEN で元に戻ります。

                moveq   #$2E,D0
                move.l  #256*65536+128,D1
                move.l  #(32-1)*65536+(16-1),D2
                trap    #15

                dc.w    $FF00

                .end
```

IOCSサンプル13

```
*
*
*       ＩＰＬの比較
*
*

*       使用方法
*               実行すると、ドライブ０とドライブ１のＩＰＬを比較します。

                moveq   #$46,D0                 * ドライブ０のＩＰＬを読み込む
                move.w  #$9070,D1
                move.l  #$03_00_00_01,D2
                move.l  #$400,D3
                lea     buffer(PC),A1
                trap    #15
                andi.l  #$FA_FF_FF_00,D0
                bne     error

                moveq   #$41,D0                 * ドライブ１のＩＰＬと比較する
                move.w  #$9170,D1
                move.l  #$03_00_00_01,D2
                move.l  #$400,D3
                lea     buffer(PC),A1
                trap    #15
                btst.l  #3+8,D0                 * 同じか？
                beq     notequal

                pea     equalmes(PC)
                dc.w    $FF09
                addq.l  #4,SP
                dc.w    $FF00
```

```
notequal
                pea     notequalmes(PC)
                dc.w    $FF09
                addq.l  #4,SP
                dc.w    $FF00

error
                pea     errormes(PC)
                dc.w    $FF09
                addq.l  #4,SP
                dc.w    $FF00

equalmes
                dc.b    'ＩＰＬは同一です。',13,10,0
notequalmes
                dc.b    'ＩＰＬが違っています。',13,10,0
errormes
                dc.b    'Error',7,13,10,0

                .bss
buffer
                ds.b    $400

                .end
```

IOCSサンプル14

```
*
*
*       Ｈｕｍａｎ６８Ｋ物理フォーマット
*
*

*       使用方法
*               実行すると、ドライブ１をフォーマットします。
*               論理フォーマットは行いません。

                moveq   #0,D7                   * D7.b = トラック
loop2
                moveq   #0,D6                   * D6.b = サイド

loop1
                lea     formatdata(PC),A0
                movea.l A0,A1

                moveq   #8-1,D0

loop0
                move.b  D7,(A0)+                * ＩＤデーター作成
                move.b  D6,(A0)+
                addq.l  #2,A0
                dbra    D0,loop0

                moveq   #$4D,D0                 * フォーマット
                move.w  #$9170,D1
                move.l  #$03_00_00_00,D2
                move.l  D7,D5
                swap    D5
                or.l    D5,D2
                move.l  D6,D5
                lsl.l   #8,D5
                or.l    D5,D2
                move.l  #8*4,D3
                trap    #15

                andi.l  #$FF_FF_00,D0
                bne     error

                addq.b  #1,D6                   * 次のサイド
                cmpi.b  #2,D6
                bcs     loop1

                addq.b  #1,D7                   * 次のトラック
                cmpi.b  #77,D7
                bcs     loop2

                dc.w    $FF00
```

```
error
                pea     errormes(PC)
                dc.w    $FF09
                addq.l  #4,SP
                dc.w    $FF00

formatdata
                dc.b    0,0,1,3                 *フォーマットＩＤデーター
                dc.b    0,0,2,3
                dc.b    0,0,3,3
                dc.b    0,0,4,3
                dc.b    0,0,5,3
                dc.b    0,0,6,3
                dc.b    0,0,7,3
                dc.b    0,0,8,3

errormes
                dc.b    'Error',13,10,0

                .end
```

IOCSサンプル15

```
*
*
*       ディスクイジェクト
*
*

*       使用方法
*               実行すると、ドライブ０＆１のディスクをイジェクトします。
*               イジェクトが禁止されている場合はイジェクトできません。

                moveq   #$4E,D0                 * ドライブ０イジェクト
                move.w  #$9000,D1
                move.w  #1,D2
                trap    #15

                moveq   #$4E,D0                 * ドライブ１イジェクト
                move.w  #$9100,D1
                move.w  #1,D2
                trap    #15

                dc.w    $FF00

                .end
```

IOCSサンプル16

```
*
*
*       日付時刻表示
*
*

*       使用方法
*               実行すると日付などが表示されます。

                moveq   #$54,D0                 * 日付の表示
                trap    #15
                move.l  D0,D7
                move.l  D0,D1
                moveq   #$55,D0
                trap    #15
                andi.l  #$0F_FF_FF_FF,D0
                move.l  D0,D1
                moveq   #$5A,D0
                lea     prtbuf(PC),A1
                trap    #15
                moveq   #$21,D0
                lea     prtbuf(PC),A1
                trap    #15

                moveq   #$20,D0                 * 曜日の表示
                move.w  #'(',D1
                trap    #15
```

```
                rol.l   #8,D7
                andi.l  #$F,D7
                move.l  D7,D1
                lea     prtbuf(PC),A1
                moveq   #$5C,D0
                trap    #15
                moveq   #$21,D0
                lea     prtbuf(PC),A1
                trap    #15

                moveq   #$20,D0
                move.w  #')',D1
                trap    #15

                moveq   #$56,D0                 * 時刻の表示
                trap    #15
                move.l  D0,D1
                moveq   #$57,D0
                trap    #15
                andi.l  #$0F_FF_FF_FF,D0
                move.l  D0,D1
                moveq   #$5B,D0
                lea     prtbuf(PC),A1
                trap    #15
                moveq   #$21,D0
                lea     prtbuf(PC),A1
                trap    #15

                lea     crlf(PC),A1             * 改行
                moveq   #$21,D0
                trap    #15

                dc.w    $FF00
crlf
                dc.b    10,13,0
prtbuf
                ds.b    32

                .end
```

IOCSサンプル17

```
*
*
*       ＡＤＰＣＭ録音&再生
*

*       使用方法
*               実行すると、まず録音をします。その後で、録音したものを
*               各周波数で再生します。

                lea     startmes(PC),A1         * キー入力待ち
                moveq   #$21,D0
                trap    #15
wait
                moveq   #0,D0
                trap    #15
                cmpi.b  #' ',D0
                bne     wait

                moveq   #$61,D0                 * 録音
                move.w  #3*256+3,D1
                lea     buffer(PC),A1
                move.l  #65536,D2
                trap    #15

wait2
                moveq   #$66,D0                 * 録音終了まで待つ
                trap    #15
                cmpi.b  #4,D0
                beq     wait2

                lea     repeatmes(PC),A1        * キー入力待ち
                moveq   #$21,D0
                trap    #15
wait3
                moveq   #0,D0
```

```
		trap	#15
		cmpi.b	#' ',D0
		bne	wait3

		moveq	#0,D3			* D3=再生周波数
loop
		move.w	D3,D1
		lsl.w	#8,D1
		move.b	#3,D1
		move.l	#65536,D2
		lea	buffer(PC),A1
		moveq	#$60,D0
		trap	#15
wait4
		moveq	#$66,D0			* 再生終了まで待つ
		trap	#15
		cmpi.b	#2,D0
		beq	wait4

		addq.w	#1,D3
		cmpi.w	#5,D3
		bcs	loop

		dc.w	$FF00

startmes
		dc.b	'スペースキーで録音開始',10,13,0
repeatmes
		dc.b	'スペースキーで再生開始',10,13,0
		.bss
buffer
		ds.b	65536

		.end
```

IOCSサンプル18

```
*
*
*	ＯＰＭによる演奏
*

*	使用方法
*		実行すると、ＯＰＭを使った演奏を行います。
*		ＯＰＭＤＲＶ．Ｘなどは登録しない状態で実行してください。

		lea	initdata(PC),A1		* ＯＰＭのレジスタを設定する
loop
		move.b	(A1)+,D1
		beq	play
		move.b	(A1)+,D2
		moveq	#$68,D0
		trap	#15
		bra	loop

play
		lea	musicdata(PC),A1	* 演奏
nextnote
		move.b	(A1)+,D3		* D3=音程
		beq	musstop
		move.b	(A1)+,D4		* D4=音長
		moveq	#$68,D0
		move.b	#$28,D1
		move.b	D3,D2
		trap	#15
		moveq	#$68,D0
		move.b	#8,D1
		move.b	#%1111_000,D2
		trap	#15

		move.l	#20000,D0
wait2
		subq.l	#1,D0
		bne	wait2
```

```
		moveq	#$68,D0
		move.b	#8,D1
		move.b	#%0000_000,D2
		trap	#15

		ext.w	D4
		mulu	#10000,D4
wait
		subq.l	#1,D4
		bne	wait
		bra	nextnote

musstop
		dc.w	$FF00			* 終了

initdata
		dc.b	$20,%11_000_100
		dc.b	$40,%010_1000
		dc.b	$48,%000_0100
		dc.b	$50,%110_1000
		dc.b	$58,%000_0100
		dc.b	$60,31
		dc.b	$68,31
		dc.b	$70,0
		dc.b	$78,0
		dc.b	$80,31
		dc.b	$88,31
		dc.b	$90,31
		dc.b	$98,31
		dc.b	$A0,7
		dc.b	$A8,7
		dc.b	$B0,3
		dc.b	$B8,3
		dc.b	$C0,1
		dc.b	$C8,1
		dc.b	$D0,3
		dc.b	$D8,3
		dc.b	$E0,10*16+3
		dc.b	$E8,10*16+3
		dc.b	$F0,5*16+3
		dc.b	$F8,5*16+3
		dc.b	0

musicdata
		dc.b	$41,20
		dc.b	$3C,20
		dc.b	$3E,20
		dc.b	$35,60
		dc.b	$35,20
		dc.b	$3E,20
		dc.b	$41,20
		dc.b	$3C,60
		dc.b	0

		.end
```

IOCSサンプル19

```
*
*
*	割り込み処理
*

*	使用方法
*		実行すると、'X'を２４００文字表示します。その間に割り込みで
*		' 'を表示させます。

		moveq	#$6D,D0			* 割り込みスタート
		clr.w	D1
		lea	jobadd(PC),A1
		trap	#15

		move.w	#2400-1,D7		* X 表示
loop
		move.w	#'X',D1
		moveq	#$20,D0
		trap	#15
		dbra	D7,loop
```

```
	moveq	#$6D,D0		* 割り込み終了
	clr.l	A1
	trap	#15

	dc.w	$FF00

jobadd
	movem.l	D0/D1,-(SP)	* 割り込みサブルーチン
	moveq	#$28,D0
	clr.w	D1
	trap	#15
	movem.l	(SP)+,D0/D1
	rte

	.end
```

IOCSサンプル20

```
*
*
*	マウスカーソル表示
*
*	使用方法
*		実行すると、マウスカーソルが表示されます。左ボタンを押すとドットを打ちます。
*		右ボタンを押すと終了します。

	moveq	#$70,D0		* マウス初期化
	trap	#15

	move.w	#0,D1
	lea	pattern(PC),A1
loop
	moveq	#$7A,D0		* マウスカーソルパターンを定義
	trap	#15
	adda.l	#4+64,A1
	addq.w	#1,D1
	cmpi.w	#3,D1
	bcs	loop

	moveq	#$77,D0		* マウスカーソル移動範囲を指定
	move.l	#100*65536+100,D1
	move.l	#668*65536+412,D2
	trap	#15

	moveq	#$76,D0		* マウスカーソルの位置を設定
	move.l	#100*65536+100,D1
	trap	#15

	moveq	#$71,D0		* マウスカーソル表示
	trap	#15
	moveq	#$7C,D0
	lea	putpat(PC),A1
	trap	#15
wait
	moveq	#$74,D0		* マウスボタンチェック
	trap	#15
	cmpi.b	#$FF,D0
	beq	quit
	andi.w	#$FF00,D0
	beq	wait

	moveq	#$75,D0		* 座標を調べる
	trap	#15
	move.w	D0,D2
	swap	D0
	move.w	D0,D1
	moveq	#$1B,D0		* 点を打つ
	lea	dot(PC),A1
	trap	#15

	bra	wait
quit
	moveq	#$72,D0		* マウスカーソルを消す
```

```
		trap	#15
		dc.w	$FF00

putpat
		dc.w	0,1,2,-1

pattern
		dc.w	8,8
		dc.w	%0000_0011_1100_0000
		dc.w	%0000_0011_1100_0000
		dc.w	%0000_0011_1100_0000
		dc.w	%0001_1111_1111_0000
		dc.w	%0001_1111_1111_0000
		dc.w	%0001_1111_1111_0000
		dc.w	%1111_1111_1111_1111
		dc.w	%1111_1111_1111_1111
		dc.w	%1111_1111_1111_1111
		dc.w	%1111_1111_1111_1111
		dc.w	%0001_1111_1111_1000
		dc.w	%0001_1111_1111_1000
		dc.w	%0001_1111_1111_1000
		dc.w	%0000_0011_1100_0000
		dc.w	%0000_0011_1100_0000
		dc.w	%0000_0011_1100_0000

		dc.w	%0000_0000_0000_0000
		dc.w	%0111_1000_0001_1110
		dc.w	%0100_0000_0000_0010
		dc.w	%0100_0000_0000_0010
		dc.w	%0100_0000_0000_0010
		dc.w	%0000_0000_0000_0000
		dc.w	%0000_0000_0000_0000
		dc.w	%0000_0000_0000_0000
		dc.w	%0000_0000_0000_0000
		dc.w	%0000_0000_0000_0000
		dc.w	%0000_0000_0000_0000
		dc.w	%0100_0000_0000_0010
		dc.w	%0100_0000_0000_0010
		dc.w	%0100_0000_0000_0010
		dc.w	%0111_1000_0001_1110
		dc.w	%0000_0000_0000_0000

		dc.w	8,8
		dc.w	%1111_1111_1111_1111
		dc.w	%1000_0001_1000_0001
		dc.w	%1000_0001_1000_0001
		dc.w	%1000_0001_1000_0001
		dc.w	%1000_1111_1111_0001
		dc.w	%1000_1111_1111_0001
		dc.w	%1000_1111_1111_0001
		dc.w	%1111_1111_1111_1111
		dc.w	%1111_1111_1111_1111
		dc.w	%1000_1111_1111_0001
		dc.w	%1000_1111_1111_0001
		dc.w	%1000_1111_1111_0001
		dc.w	%1000_0001_1000_0001
		dc.w	%1000_0001_1000_0001
		dc.w	%1000_0001_1000_0001
		dc.w	%1111_1111_1111_1111

		dc.w	%0000_0000_0000_0000
		dc.w	%0000_0000_0000_0000
		dc.w	%0011_1100_0011_1100
		dc.w	%0010_0000_0000_0100
		dc.w	%0010_0000_0000_0100
		dc.w	%0010_0000_0000_0100
		dc.w	%0000_0000_0000_0000
		dc.w	%0000_0000_0000_0000
		dc.w	%0000_0000_0000_0000
		dc.w	%0000_0000_0000_0000
		dc.w	%0010_0000_0000_0100
		dc.w	%0010_0000_0000_0100
		dc.w	%0010_0000_0000_0100
		dc.w	%0011_1100_0011_1100
		dc.w	%0000_0000_0000_0000
		dc.w	%0000_0000_0000_0000

		dc.w	8,8
		dc.w	%1111_1111_1111_1111
```

```
        dc.w    %1111_1111_1111_1111
        dc.w    %1100_0000_0000_0011
        dc.w    %1100_0000_0000_0011
        dc.w    %1100_0000_0000_0011
        dc.w    %1100_0111_1110_0011
        dc.w    %1100_0111_1110_0011
        dc.w    %1100_0111_1110_0011
        dc.w    %1100_0111_1110_0011
        dc.w    %1100_0111_1110_0011
        dc.w    %1100_0111_1110_0011
        dc.w    %1100_0000_0000_0011
        dc.w    %1100_0000_0000_0011
        dc.w    %1100_0000_0000_0011
        dc.w    %1111_1111_1111_1111
        dc.w    %1111_1111_1111_1111

        dc.w    %0000_0000_0000_0000
        dc.w    %0000_0000_0000_0000
        dc.w    %0000_0000_0000_0000
        dc.w    %0001_1110_0111_1000
        dc.w    %0001_0000_0000_1000
        dc.w    %0001_0000_0000_1000
        dc.w    %0001_0000_0000_1000
        dc.w    %0000_0000_0000_0000
        dc.w    %0000_0000_0000_0000
        dc.w    %0001_0000_0000_1000
        dc.w    %0001_0000_0000_1000
        dc.w    %0001_0000_0000_1000
        dc.w    %0001_1110_0111_1000
        dc.w    %0000_0000_0000_0000
        dc.w    %0000_0000_0000_0000
        dc.w    %0000_0000_0000_0000
dot
        dc.w    1,1
        dc.b    $80

        .end
```

IOCSサンプル21

```
*
*
*       テキストVRAM転送によるスクロール
*
*       使用方法
*           実行すると、テキスト画面の一部がスクロールします。

        moveq   #$8A,D0                 * プレーン0スクロール
        move.b  #%1010,D1
        move.l  #20480-128,D2
        lea     $E05000-1-128,A1
        lea     $E05000-1,A2
        trap    #15

        moveq   #$8A,D0                 * プレーン1スクロール
        move.b  #%1010,D1
        move.l  #20480-128,D2
        lea     $E25000-1-128,A1
        lea     $E25000-1,A2
        trap    #15

        dc.w    $FF00

        .end
```

IOCSサンプル22

```
*
*
*       グラフィックデモ
*

*       使用方法
*               実行すると、ＩＯＣＳ＄９８、＄９９を使った表示を行います。
*               グラフィック画面を使用します。
*               SCREEN で元に戻ります。

                moveq   #$10,D0                 * 画面モード設定
                move.w  #10,D1
                trap    #15
                moveq   #$90,D0
                trap    #15

                clr.w   D1                      * ＩＯＣＳ＄９８による書き込み
                move.w  #230,D2
                clr.w   D3
                lea     grdata(PC),A1

putloop0
                moveq   #$98,D0
                trap    #15

                addi.w  #9,D1
                subi.w  #3,D2

                cmpi.w  #230,D1
                bcs     putloop0

                clr.w   D1                      * ＩＯＣＳ＄９９による書き込み
                move.w  #180,D2
                lea     grdata(PC),A1

putloop1
                moveq   #$99,D0
                trap    #15

                addi.w  #9,D1
                subi.w  #3,D2

                cmpi.w  #230,D1
                bcs     putloop1

                dc.w    $FF00

grdata
                dc.w    16,16,$FF
                dc.b    $FF,$FF,$FF,$FF,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
                dc.b    $FF,$D5,$D5,$D5,$FF,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
                dc.b    $FF,$D5,$D5,$D5,$D5,$FF,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
                dc.b    $FF,$D5,$D5,$D5,$D5,$D5,$FF,$00,$00,$00,$00,$00,$00,$00,$00,$00
                dc.b    $00,$FF,$D5,$D5,$D5,$D5,$D5,$FF,$00,$00,$FF,$FF,$FF,$FF,$FF,$FF
                dc.b    $00,$00,$FF,$D5,$D5,$D5,$D5,$D5,$FF,$00,$FF,$D5,$D5,$D5,$D5,$FF
                dc.b    $00,$00,$00,$FF,$D5,$D5,$D5,$D5,$D5,$FF,$FF,$D5,$D5,$D5,$D5,$FF
                dc.b    $00,$00,$00,$00,$FF,$D5,$D5,$D5,$D5,$D5,$FF,$D5,$D5,$D5,$D5,$FF
                dc.b    $00,$00,$00,$00,$00,$FF,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$FF
                dc.b    $00,$00,$00,$00,$00,$00,$FF,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$FF
                dc.b    $00,$00,$00,$00,$FF,$FF,$FF,$FF,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$FF
                dc.b    $00,$00,$00,$00,$FF,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$FF
                dc.b    $00,$00,$00,$00,$FF,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$FF
                dc.b    $00,$00,$00,$00,$FF,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$FF
                dc.b    $00,$00,$00,$00,$FF,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$D5,$FF
                dc.b    $00,$00,$00,$00,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF

                .end
```

```
*
*
*       グラフィックデモ
*

*       使用方法
*           実行すると、ＩＯＣＳ＄９Ａ、＄９Ｂ、＄９Ｃを使った表示を行います。
*           グラフィック画面を使用します。
*           SCREEN で元に戻ります。

            moveq   #$10,D0                 * 画面モード設定
            move.w  #10,D1
            trap    #15
            moveq   #$90,D0
            trap    #15

            moveq   #$19,D0                 * フォントデーターを読み出す
            move.l  #$C*65536+'漢',D1
            lea     buffer(PC),A1
            trap    #15

            moveq   #$95,D0                 * カラーコードを指定
            move.w  #$FF,D1
            trap    #15

            clr.w   D1
            clr.w   D2
            lea     buffer(PC),A1
putloop0
            moveq   #$9A,D0                 * ＩＯＣＳ＄９Ａによる書き込み
            trap    #15

            addi.w  #15,D1
            addq.w  #4,D2
            cmpi.w  #230,D1
            bcs     putloop0

            clr.w   D1
            move.w  #$30,D3
putloop1
            moveq   #$9B,D0                 * ＩＯＣＳ＄９Ｂによる書き込み
            trap    #15

            addi.w  #15,D1
            addq.w  #4,D2
            cmpi.w  #230,D1
            bcs     putloop1

            clr.w   D1
            move.w  #3,D3
            move.w  #2,D4
putloop2
            moveq   #$9C,D0                 * ＩＯＣＳ＄９Ｃによる書き込み
            trap    #15

            addi.w  #30,D1
            addq.w  #4,D2
            cmpi.w  #200,D1
            bcs     putloop2

            dc.w    $FF00

buffer
            ds.w    2
            ds.b    72

            .end
```

IOCSサンプル24

```
*
*
*       ＩＯＣＳ＄Ｂ０番台によるグラフィックデモ
*
*       使用方法
*               実行すると、ＩＯＣＳ＄Ｂ０番台を使ったデモを行います。
*               何かキーを押すと終了します。

                moveq   #$10,D0                 * 画面初期化
                moveq   #4,D1                   * ５１２×５１２×４枚
                trap    #15

                moveq   #$90,D0                 * グラフィック表示
                trap    #15

                moveq   #$B1,D0                 * ０ページを指定
                moveq   #0,D1
                trap    #15
                lea     circle(PC),A1           * 円を描く
                moveq   #$BB,D0
                trap    #15
                lea     paint(PC),A1            * 円を塗り潰す
                moveq   #$BC,D0
                trap    #15

                moveq   #$B1,D0                 * １ページを指定
                moveq   #1,D1
                trap    #15
                lea     fill(PC),A1             * 塗り潰しボックスを描く
                moveq   #$BA,D0
                trap    #15

                moveq   #$B1,D0                 * ２ページを指定
                moveq   #2,D1
                trap    #15
                lea     box(PC),A1              * ボックスを描く
                moveq   #$B9,D0
                trap    #15
                lea     line(PC),A1             * ラインを描く
                moveq   #$B8,D0
                trap    #15

                moveq   #$B1,D0                 * ３ページを指定
                moveq   #3,D1
                trap    #15
                lea     symbol(PC),A1           * 文字を描く
                moveq   #$BD,D0
                trap    #15

                clr.w   p0x                     * 表示位置初期データー
                clr.w   p1x
                clr.w   p2y
                clr.w   p3y

loop
                moveq   #1,D0                   * キー入力チェック
                trap    #15
                tst.l   D0
                bne     return

                moveq   #$B3,D0                 * 表示位置変更
                moveq   #1,D1
                move.w  p0x(PC),D2
                andi.w  #511,D2
                clr.w   D3
                trap    #15
                moveq   #$B3,D0
                moveq   #2,D1
                move.w  p1x(PC),D2
                andi.w  #511,D2
                clr.w   D3
                trap    #15
                moveq   #$B3,D0
                moveq   #4,D1
                move.w  p2y(PC),D3
                andi.w  #511,D3
```

```
        clr.w   D2
        trap    #15
        moveq   #$B3,D0
        moveq   #8,D1
        move.w  p3y(PC),D3
        andi.w  #511,D3
        clr.w   D2
        trap    #15

        addi.w  #1,p0x                  * 表示位置移動
        subi.w  #1,p1x
        addi.w  #1,p2y
        subi.w  #1,p3y

        move.l  #5000,D0
wait
        subq.l  #1,D0
        bne     wait

        bra     loop

return
        dc.w    $FF00

p0x
        ds.w    1
p1x
        ds.w    1
p2y
        ds.w    1
p3y
        ds.w    1

line
        dc.w    256,0,256,511,11,%1110011010110011
box
        dc.w    64,64,448,448,9,%1111000011110000
fill
        dc.w    128,128,384,384,7
circle
        dc.w    256,256,128,5,0,360,256
paint
        dc.w    256,256,5
        dc.l    work0,work1
symbol
        dc.w    128,256
        dc.l    string
        dc.b    1,2
        dc.w    13
        dc.b    2,0

string
        dc.b    'AABBC',0

        .even

work0
        ds.b    256
work1

        .end
```

IOCSサンプル25

```
*
*
*       スプライトデモ
*

*       使用方法
*               実行すると、スプライトIOCSを使ったデモを行います。
*               何かキーを押すと終了します。

        moveq   #$10,D0                 * 画面モード設定
        moveq   #2,D1                   * 256×256表示
        trap    #15

        moveq   #$C0,D0                 * スプライト初期化
```

```
          trap    #15

          moveq   #$C1,D0             * スプライト画面表示
          trap    #15

          moveq   #$C4,D0             * ＰＣＧパターン定義
          moveq   #0,D1
          moveq   #1,D2
          lea     patdat(PC),A1
          trap    #15

          moveq   #0,D7               * パレット定義
          lea     paldat(PC),A0
loop
          moveq   #$CF,D0
          move.l  D7,D1
          moveq   #1,D2
          moveq   #0,D3
          move.w  (A0)+,D3
          trap    #15
          addq.l  #1,D7
          cmpi.b  #9,D7
          bcs     loop

          move.l  #3,D6               * 初期位置設定
          move.l  #-4,D7
          move.l  #16,D2
          move.l  #16,D3

mainloop
          moveq   #0,D1               * 表示
          move.l  #$100,D4
          move.l  #3,D5
          moveq   #$C6,D0
          trap    #15

          add.l   D6,D2               * 座標移動
          cmpi.l  #16,D2
          bcc     l0
          neg.l   D6
          add.l   D6,D2
l0
          cmpi.l  #240,D2
          bcs     l1
          neg.l   D6
          add.l   D6,D2
l1
          add.l   D7,D3
          cmpi.l  #16,D3
          bcc     l2
          neg.l   D7
          add.l   D7,D3
l2
          cmpi.l  #240,D3
          bcs     l3
          neg.l   D7
          add.l   D7,D3

l3
          moveq   #1,D0               * キー入力チェック
          trap    #15
          tst.l   D0
          beq     mainloop

          dc.w    $FF00


patdat
          dc.l    $00000011
          dc.l    $00011122
          dc.l    $00122233
          dc.l    $01223344
          dc.l    $01233455
          dc.l    $01234566
          dc.l    $12345677
          dc.l    $12345677

          dc.l    $12345677
          dc.l    $12345677
```

```
        dc.l    $01234566
        dc.l    $01233455
        dc.l    $01223344
        dc.l    $00122233
        dc.l    $00011122
        dc.l    $00000011

        dc.l    $11000000
        dc.l    $22111000
        dc.l    $33222100
        dc.l    $88332210
        dc.l    $55833210
        dc.l    $66543210
        dc.l    $77684321
        dc.l    $77654321

        dc.l    $77654321
        dc.l    $77654321
        dc.l    $66543210
        dc.l    $55433210
        dc.l    $44332210
        dc.l    $33222100
        dc.l    $22111000
        dc.l    $11000000

paldat
        dc.w    %00000_00000_00000_0
        dc.w    %00010_00001_00100_1
        dc.w    %00100_00010_01000_1
        dc.w    %00110_00011_01100_1
        dc.w    %01000_00100_10000_1
        dc.w    %01010_00101_10100_1
        dc.w    %01100_00110_11000_1
        dc.w    %01110_00111_11100_1
        dc.w    %11111_11111_11111_1

        .end
```

IOCSサンプル26

```
*
*
*       バックグラウンド表示
*
*       使用方法
*               実行すると、バックグラウンドＩＯＣＳを使用した表示を行います。
*               SCREEN で元に戻ります。

        moveq   #$10,D0                 * 画面モード設定
        moveq   #2,D1                   * ２５６×２５６表示
        trap    #15

        moveq   #$C0,D0                 * スプライト初期化
        trap    #15

        moveq   #$C1,D0                 * スプライト画面表示
        trap    #15

        moveq   #$C4,D0                 * ＰＣＧパターン定義
        moveq   #0,D1
        moveq   #0,D2
        lea     patdat(PC),A1
        trap    #15

        moveq   #$CC,D0                 * ＢＧテキストページを０のＰＣＧで埋める
        moveq   #0,D1
        move.l  #$100,D2
        trap    #15

        moveq   #$CA,D0                 * ＢＧ表示
        moveq   #0,D1
        moveq   #0,D2
        moveq   #1,D3
        trap    #15

        dc.w    $FF00
```

```
patdat
		dc.l	$F000000F
		dc.l	$0F0000F0
		dc.l	$00F00F00
		dc.l	$000FF000
		dc.l	$000FF000
		dc.l	$00F00F00
		dc.l	$0F0000F0
		dc.l	$F000000F

		.end
```

IOCSサンプル27

```
*
*
*	テキストＶＲＡＭラスターコピー
*

*	使用方法
*		実行すると、ラスターコピーを使ったスクロールを行います。
*		SCREEEN で元に戻ります。

		moveq	#$D6,D0			* テキスト画面にボックスを描く
		lea	box(PC),A1
		trap	#15

		moveq	#$D7,D0			* テキスト画面に塗り潰しボックスを描く
		lea	fill(PC),A1
		trap	#15

		moveq	#$D8,D0			* テキスト画面を反転する
		lea	rev(PC),A1
		trap	#15

		moveq	#$DF,D0			* コピー
		move.l	#0*256+32,D1
		move.l	#0*256+32,D2
		move.l	#%11,D3
		trap	#15

		dc.w	$FF00

box
		dc.w	0,128,64,640,256,%1111000011110000
fill
		dc.w	0,256,128,384,128,%1100110011001100
rev
		dc.w	0,384,0,192,512

		.end
```

IOCSサンプル28

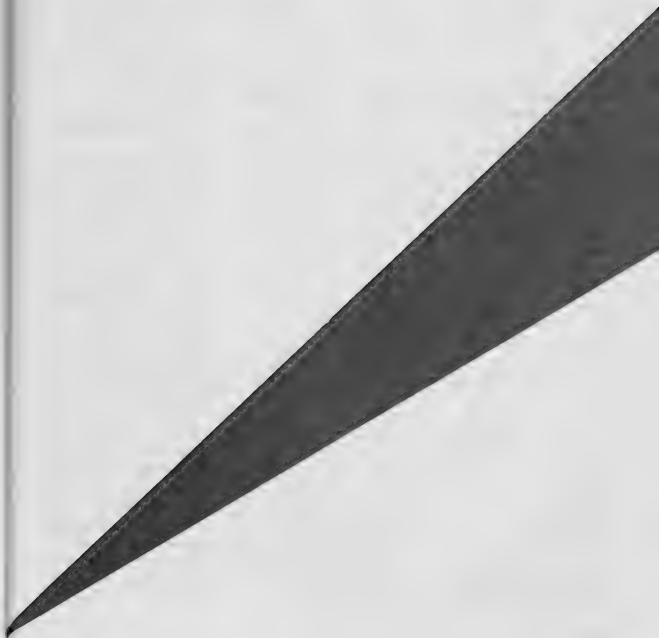
```
*
*
*	ＯＰＭＤＲＶによる演奏
*

*	使用方法
*		実行すると、ＯＰＭＤＲＶによって拡張されるＩＯＣＳを使用した
*		演奏を行います。

		moveq	#$F0,D0			* 初期化
		moveq	#0,D1
		trap	#15

		moveq	#$F0,D0			* トラックバッファの確保
		moveq	#1,D1
		move.l	#1*65536+10000,D2
		trap	#15

		moveq	#$F0,D0			* 演奏データーセット
		moveq	#6,D1
		moveq	#1,D2
		lea	music(PC),A1
		trap	#15
```

```
        moveq   #$F0,D0
        moveq   #8,D1
        moveq   #1,D2
        trap    #15
loop
        moveq   #$F0,D0
        moveq   #9,D1
        moveq   #1,D2
        trap    #15

        tst.l   D0
        bne     loop

        dc.w    $FF00
music
        dc.b    'V1505D+8C+804F+405F+8R8F+8R8'
        dc.b    'D+8C+804F+405F+8R8F+8R8'
        dc.b    'D+8C+804F+405F+404D+405F+404C+405F8R8F8R8',0

        .end
```

IOCSサンプル29

```
*
*
*       AJOYによるマウス<ーーアナログジョイスティック
*

*       使用方法
*               実行すると、アナログジョイスティックでマウス入力ができるようになります。
*               アナログジョイスティックはポート1にセットしてください。
*               もう一度実行するともとに戻ります。

portadd         equ     $E9A001
reqadd          equ     $E9A005
reqbit          equ     4

                bra     next
check
                dc.b    'ｱﾅｼﾞｮｲﾏｳｽ!'
trueadd
                ds.l    1
next
                movea.l (A0),A1
                adda.l  #$100+2,A1
                lea     check(PC),A2

                moveq   #10-1,D0
loop0
                cmpm.b  (A1)+,(A2)+
                bne     progset
                dbf     D0,loop0
                bra     remove
progset
                moveq   #$84,D0                 * 処理先を変更する
                lea     $934,A1
                trap    #15
                move.l  D0,(trueadd)

                moveq   #$88,D0
                move.l  #intadd,D1
                lea     $934,A1
                trap    #15

                moveq   #$F2,D0                 * アナログモード
                moveq   #1,D1
                moveq   #1,D2
                trap    #15

                clr.w   -(SP)
                move.l  #1000,-(SP)
                dc.w    $FF31
```

```
remove                                          * 切り放し
        movea.l (A0),A1
        adda.l  #$10,A1
        move.l  A1,-(SP)
        dc.w    $FF49
        addq.l  #4,SP

        adda.l  #$FC,A1
        move.l  (A1),D1
        lea     $934,A1
        moveq   #$88,D0
        trap    #15

        dc.w    $FF00

intadd                                          * データーをすり替え
        movem.l D0-D1/A0/A1,-(SP)
        movea.l A1,A0

        lea     buffer(PC),A1
        moveq   #$F2,D0
        moveq   #0,D1
        trap    #15
        tst.l   D0
        bmi     error

        clr.b   D0
        btst.b  #3,8(A1)
        bne     line0
        bset.l  #0,D0
line0
        btst.b  #2,8(A1)
        bne     line1
        bset    #1,D0
line1
        move.b  D0,(A0)+

        move.w  2(A1),D0
        subi.w  #128,D0
        bsr     center
        move.b  D0,(A0)+

        move.w  (A1),D0
        subi.w  #128,D0
        bsr     center
        move.b  D0,(A0)+

error
return
        movem.l (SP)+,D0-D1/A0/A1

        move.l  trueadd(PC),-(SP)
        rts

center                                          * あそびを持たせる
        asr.b   #3,D0
        tst.b   D0
        bmi     line2
        subq.b  #3,D0
        bpl     line3
        clr.b   D0
        bra     line3
line2
        addq.b  #3,D0
        bmi     line3
        clr.b   D0
line3
        rts

buffer
        ds.b    10

        .end
```

付録

Human68k　2HD ディスクマップ

Human68kの2HDディスクは1セクタ1024バイト，1トラック16セクタ（サーフェス0&1），となっています．セクタの順番はトラック0サーフェス0のセクタ1が先頭で0番セクタ，以降はセクタ8が7番セクタ，同じトラックのサーフェス1のセクタが8番から15番セクタ，トラック1サーフェス0のセクタ1が16番セクタとなります．

●セクタ番号

0	IPL プログラムが書き込まれている
1～2	第1ＦＡＴ
3～4	第2ＦＡＴ（未使用）
5～＄Ａ	ルートディレクトリ
＄Ｂ～	データーエリア

● IPL プログラム

$2000番地にロードされて実行されます．標準のIPLは実行されると「HUMAN. SYS」をロードして起動します．

●第１ＦＡＴ

ファイルがデーターエリアのどこに書き込まれているか，また，そのデーターのつながりを示します．データーエリアの１セクタにつき１つのFATが対応していて，FATの値によって対応したセクタの内容が分かります．

FATは次のように並んでいます．先頭の２つ分はシステムによって値が決められています．３つ目のFATがデーターエリアの最初のセクタに対応しています．

FAT の並び方

３バイトで２つのＦＡＴを表しています．FATは先頭の２つを含めて順に０からの番号がついています．

	+0	+1	+2	+3	+4	+5	+6	+7
	F9	FF	FF	03	40	00	FF	0F
	先頭2つ		22	32	33	44	54	FATナンバー
			ML	LH	HM	ML	LH	16進数の場合の位（High-Middle-Low）

↓

FAT 2 = $003

FAT 3 = $004

FAT 4 = $FFF　　となります．

FAT の値の意味

$000	未使用セクタ
$002～$FF6	次のＦＡＴナンバー（ＦＡＴは先頭の２つを含めて順に０からの番号がついている）
$FF7	使用不能セクタ
$FFF	データブロックの終わり

●ルートディレクトリ

どのようなファイルが書き込まれているかを示します．１つのファイル（ディレクトリ&ボリュームネームも含む）につき32バイトが割り当てられ，次のような構成になっています．

+0.b ～+7.b　ファイル名
　先頭が０の場合はディレクトリの終わりを示す
　先頭が$E5の場合はデリートされたファイルを示す($E5で始まるファイルは５となる)
　余った部分には$20がセットされる

+8.b - +$A.b　拡張子
　余った部分には$20がセットされる

+$B.b　属性

bit 0	読み込み専用
bit 1	不可視
bit 2	システム
bit 3	ボリュームネーム
bit 4	ディレクトリ名
bit 5	通常のファイル

bit 3 から bit 5 はそれらのうちどれか一つが１になる

+$C.b ～+$15.b　ファイル名の残り
　余った部分には$20がセットされる

+$16.w　時刻
　bit15～bit 11時

	bit10～bit 5 分
	bit4 ～bit 0 秒×2
+$18.w	日付
	bit15～bit 9 年
	bit8 ～bit 5 月
	bit4 ～bit 0 日
+$1A.w	最初のＦＡＴナンバー
+$1C.l	ファイルサイズ

※なお，ワード，ロングワードのパラメーターは上位と下位が８ビット単位で逆に並んでいる

●サブディレクトリ

サブディレクトリはデーターエリアに置かれ，構成はルートディレクトリと同じです．

I/O ポートアドレス

「W」は書き込み専用ポート，「R」は読み出し専用ポート，無印は読み書きポート

▶グラフィック VRAM

$C00000～$C7FFFF　グラフィックVRAM（512×512ドット65536色１画面モード時）
$C00000～$CFFFFF　（512×512ドット256色２画面モード時）
$C00000～$DFFFFF　（512×512ドット16色４画面モード時）
$C00000～$DFFFFF　（1024×1024ドット16色１画面モード時）
１ワードが１ドットに対応する

▶テキスト VRAM

$E00000～$E7FFFF　テキストVRAM

▶ CRTC

$E80000.w	W	水平トータル
$E80002.w	W	水平同期終了位置
$E80004.w	W	水平表示開始位置
$E80006.w	W	水平表示終了位置
$E80008.w	W	垂直トータル
$E8000A.w	W	垂直同期終了位置
$E8000C.w	W	垂直表示開始位置
$E8000E.w	W	垂直表示終了位置
$E80010.w	W	外部同期水平アジャスト
$E80012.w	W	ラスター割り込み位置
$E80014.w	W	テキスト画面Ｘ方向表示位置
$E80016.w	W	テキスト画面Ｙ方向表示位置
$E80018.w	W	グラフィック画面スクリーン０Ｘ方向表示位置
$E8001A.w	W	グラフィック画面スクリーン０Ｙ方向表示位置
$E8001C.w	W	グラフィック画面スクリーン１Ｘ方向表示位置
$E8001E.w	W	グラフィック画面スクリーン１Ｙ方向表示位置
$E80020.w	W	グラフィック画面スクリーン２Ｘ方向表示位置
$E80022.w	W	グラフィック画面スクリーン２Ｙ方向表示位置
$E80024.w	W	グラフィック画面スクリーン３Ｘ方向表示位置
$E80026.w	W	グラフィック画面スクリーン３Ｙ方向表示位置
$E80028.w		グラフィックVRAMメモリモード，表示モード
$E8002A.w		テキストアクセスモード，グラフィック高速クリアプレーンセレクト
$E8002C.w	W	ソース&ディスティネーションラスター
$E8002E.w	W	テキストビットマスクレジスタ
$E80480.w		CRTC動作設定

▶グラフィックパレット

$E82000.w	グラフィックパレット０
※	
$E8201E.w	グラフィックパレット15
$E82020.w	グラフィックパレット16
※	
$E821FE.w	グラフィックパレット255

▶テキスト&スプライトパレット

$E82200.w	テキストパレット0（スプライトパレットブロック0パレットコード0）
※	
$E8221E.w	テキストパレット15（スプライトパレットブロック0パレットコード15）
$E82220.w	スプライトパレットブロック1パレットコード0
※	
$E8223E.w	スプライトパレットブロック1パレットコード15
※	
$E823E0.w	スプライトパレットブロック15パレットコード0
※	
$E823FE.w	スプライトパレットブロック15パレットコード15

▶ビデオコントローラー

$E82400.w	グラフィックVRAMメモリモード
$E82500.w	プライオリティー設定
$E82600.w	特殊モード設定

▶DMAC

チャンネル0

$E84000.b	チャンネルステータスレジスタ
$E84001.b R	チャンネルエラーレジスタ
$E84004.b	デバイスコントロールレジスタ
$E84005.b	オペレーションコントロールレジスタ
$E84006.b	シーケンスコントロールレジスタ
$E84007.b	チャンネルコントロールレジスタ
$E8400A.w	メモリ転送カウンタ
$E8400C.l	メモリアドレスレジスタ
$E84014.l	デバイスアドレスレジスタ
$E8401A.w	ベース転送カウンタ
$E8401C.l	ベースアドレスレジスタ
$E84025.b	ノーマルインタラプトベクタ
$E84027.b	エラーインタラプトベクタ
$E84029.b	メモリファンクションコード
$E8402D.b	チャンネルプライオリティーレジスタ
$E84031.b	デバイスファンクションコード
$E84039.b	ベースファンクションコード

チャンネル1はチャンネル0のアドレス＋$40

チャンネル2はチャンネル0のアドレス＋$80

チャンネル3はチャンネル0のアドレス＋$C0

$E840FF.b	ジェネラルコントロールレジスタ

▶スーパーバイザー・エリアセット

$E86001.b W	スーパーバイザーエリア設定

MFP

$E88001.b R	GPIPデーターレジスタ
$E88003.b W	アクティブエッジレジスタ
$E88005.b W	データーディレクションレジスタ
$E88007.b	割り込みイネーブルレジスタA
$E88009.b	割り込みイネーブルレジスタB
$E8800B.b	割り込みペンディングレジスタA
$E8800D.b	割り込みペンディングレジスタB
$E8800F.b	割り込みインサービスレジスタA
$E88011.b	割り込みインサービスレジスタB
$E88013.b	割り込みマスクレジスタA
$E88015.b	割り込みマスクレジスタB
$E88017.b W	ベクタレジスタ
$E88019.b W	タイマーAコントロールレジスタ
$E8801B b W	タイマーBコントロールレジスタ
$E8801D.b W	タイマーC&Dコントロールレジスタ
$E8801F.b	タイマーAデーターレジスタ
$E88021.b	タイマーBデーターレジスタ
$E88023.b	タイマーCデーターレジスタ
$E88025.b	タイマーDデーターレジスタ

$E88027.b W	同期キャラクタレジスタ
$E88029.b W	USARTコントロールレジスタ
$E8802B.b	受信ステータスレジスタ
$E8802D.b	送信ステータスレジスタ
$E8802F.b	USARTデーターレジスタ

▶RTC

$E8A001.b	1秒カウンタ・CLKOUTセレクト
$E8A003.b	10秒カウンタ・アジャスト
$E8A005.b	1分カウンタ・アラーム1分レジスタ
$E8A007.b	10分カウンタ・アラーム10分レジスタ
$E8A009.b	1時間カウンタ・アラーム1時間レジスタ
$E8A00B.b	10時間カウンタ・アラーム10時間レジスタ
$E8A00D.b	曜日カウンタ・アラーム曜日レジスタ
$E8A00F.b	1日カウンタ・アラーム1日レジスタ
$E8A011.b	10日カウンタ・アラーム10日レジスタ
$E8A013.b	1月カウンタ
$E8A015.b	10月カウンタ・12時間，24時間セレクト
$E8A017.b	1年カウンタ・閏年カウンタ
$E8A019.b	10年カウンタ
$E8A01B.b	モードレジスタ
$E8A01D.b W	テストレジスタ
$E8A01F.b W	リセットコントローラー

▶プリンタポート

$E8C001.b W	データーポート
$E8C003.b W	ストローブ
$E8C005.b R	ビジー

▶システムポート

$E8E001.b	コントラスト
$E8E003.b	ＴＶコントロール
$E8E005.b W	画像入力コントロール
$E8E007.b	LED点灯・NMIクリア・キーコントロール
$E8E00D.b W	SRAM書き込み許可
$E8E00F.b W	パワーオフコントロール

▶FM音源

$E90001.b W	レジスタナンバー
$E90003.b	データ

▶ADPCM

$E92001.b	ステータス・コマンド
$E92003.b	データレジスタ
$E9A005.b W	出力モード，周波数切り替えレジスタ

▶FDC

$E94001.b R	ステータスレジスタ
$E94003.b	データレジスタ
$E94005.b	ドライブステータス・ドライブコントロール
$E94007.b W	アクセスドライブセレクト

▶ハードディスク

$E96001.b	データ
$E96003.b	ステータス・セレクトリセット
$E96005.b W	コントローラーボードリセット
$E96007.b W	セレクトセット

▶SCC

$E98001.b	コマンドポートB
$E98003.b	データーポートB
$E98005.b	コマンドポートA
$E98007.b	データーポートA

▶ジョイスティック

$E9A001.b R	ジョイスティック0
$E9A003.b R	ジョイスティック1

▶8255

$E9A007.b W	8255コントロールワードレジスタ

ポートAは $E9A001.b
ポートBは $E9A003.b
ポートCは $E9A005.b

▶ FDC, FDD, HD, プリンター

$E9C001.b	割り込みステータス，割り込みマスク
$E9C003.b W	割り込みベクタ

スプライト

$EB0000.w	スプライト0　X座標
$EB0002.w	スプライト0　Y座標
$EB0004.w	スプライト0　コントロールレジスタ
$EB0006.w	スプライト0　プライオリティー
※	
$EB03F8.w	スプライト127　X座標
$EB03FA.w	スプライト127　Y座標
$EB03FC.w	スプライト127　コントロールレジスタ
$EB03FE.w	スプライト127　プライオリティー
$EB0800.w	BG0　X座標
$EB0802.w	BG0　Y座標
$EB0804.w	BG1　X座標
$EB0806.w	BG1　Y座標
$EB0808.w	BGコントロールレジスタ
$EB080A.w	水平トータル
$EB080C.w	水平表示
$EB080E.w	垂直表示
$EB0810.w	解像度

$EB8000～$EB807F PCG0フォント
※
$EBBF80～$EBBFFF PCG127フォント

$EBC000～$EBDFFF BGテキスト0
$EBE000～$EBFFFF BGテキスト1

ABD.X

RS232Cから入力される文字列データをバッファに保存するプログラムです．バックグラウンド機能を利用しているので，他のプログラムと平行動作します．

実際にダウンロードしている文字列は表示されませんが，現在までに受信した行数，バッファの使用割合，ABD_COREのステータス等を画面左上に表示します．

[使用法]

ABD <送信文字列>-B<size>-L<area>-V<area>-S-C-F-R

<送信文字列>: RS232Cにそのまま出力されます．末尾に改行コードは付加しません．必要な場合は，ESCMを使用してください．

また，ダブルクォートで囲むことでデリミタを含む文字も指定できますが，先頭が"／"あるいは"-"の文字列は正しく出力されません．

-B<size> : ダウンロードバッファのサイズを指定します．
単位はKB．32～255の範囲で指定します．
デフォルトは64KB

-L<area> : 指定した範囲のダウンロードバッファの内容を表示します．

ex. -L2-10 ... 2行目から10行目まで
-L-20 ... 先頭から20行目まで
-L50- ... 50行目以降
-L24 ... 24行目のみ
-L ... 全行

バッファの内容をファイルに落とす場合はリダイレクトを利用します．

-V<area> : 指定した範囲のダウンロードバッファの内容を行番号付きで表示します．

-S : ダウンロードを一時停止します．

ターミナル等を利用する場合，RS232Cの入力を奪いあうので，あらかじめこのオプションでダウンロードを停止してください．

-C　　：ダウンロードを再開します.

-F　　：ダウンロードバッファをフラッシュします.
ダウンロードが停止されていた場合,再開します.

-P　　：ABD_COREのスレッドをサスペンド/サスペンド解除します.

ABDが必要ない場合,サスペンドしておいた方がフォアグラウンドが軽くなります.

-R　　：プログラムをスレッドから削除し,メモリを開放します.

[実行ファイル作成方法]

AS ABD
LK ABD
AS ABDP
LK ABDP
BIND ABD ABDP

[注意]

1）このプログラムを実行するためには以下の条件が必要です.

★OSはHuman2.0x
バックグラウンド機能，オーバレイXファイル，コモンエリア等を使用しているため，Human1.0xでは動作しません.バージョンが適当でない場合，メッセージを表示してアボートします.

★CONFIG.SYSで"COMMON＝1"以上の設定を行っている.
設定していない場合,-L,-Vでハングアップします.

★CONFIG.SYSで"PROCESS＝"の設定を行っている
設定していない場合，メッセージを表示してアボートします.

2）このプログラムは，ダウンロードした文字列を行単位で扱います．行の終端はLFで判断していますので，行末コードがCR＋LF無手順データ以外の受信にはプログラムの変更が必要です.

ABD.S

```
*
*       Automatic Background Downloader Ver1.00
*                                    (Human2.0x Only)
*       Created 1989 by M.Yoshizawa
*
*       モジュール#0  "ABD.X" (メインモジュール)
*

                include macro.h

FEFUNC          macro   number
                dc.b    $FE
                dc.b    number
                endm

                .text
start:
                lea     $10(A0),A0         * メモリ管理ポインタ＋$10
                move.l  A0,PSP_plus_10
                move.l  A2,cmdline

                FNC     $30                * vernum
                cmp.w   #$2_00,D0          * Human2.0x?
                bcs     ver_err            *  それ以下ならばver_errへ

                move.l  #BOTTOM-start+$F0,-(SP) * 自分に必要最小限のメモリ
                move.l  PSP_plus_10,-(SP) * 現在のPSP
                FNC     $4A                * setblock
                addq.l  #8,SP
```

```
        move.l  PSP_plus_10,A0
        lea     -$10(A0),A0
        lea     $80(A0),A1          * オーバレイXファイル名を再構成
        lea     name0,A2
        move.w  #64+2-1,D0
name_loop1:
        move.b  (A1)+,(A2)+
        dbeq    D0,name_loop1

        subq.l  #1,A2
        lea     $C4(A0),A1
        move.w  #64-1,D0
name_loop2:
        move.b  (A1)+,(A2)+
        dbeq    D0,name_loop2

        pea     name                * モジュールの名前"ABDP.X"
        pea     name0               * オーバレイXファイルの名前
        move.w  #5,-(SP)            * check
        FNC     $4B                 * exec
        lea     10(SP),SP
        tst.l   D0
        bmi     module_err

        clr.l   -(SP)               * 現在の環境を引き継ぐ
        move.l  cmdline,-(SP)       * コマンドライン
        pea     name0               * オーバレイXファイル名
        move.w  D0,-(SP)            * モジュール＃1をLOAD&EXEC
        FNC     $4B                 * exec
        lea     14(SP),SP
        tst.l   D0                  * エラー？
        bmi     module_err          *   ならばmodule_errへ

        tst.w   D0                  * 非常駐終了？
        bpl     __exit

        and.l   #$FF,D0             * バッファの大きさ
        lsl.l   #8,D0
        lsl.l   #2,D0               * KB単位に直す
        move.l  D0,memlen

        move.l  D0,-(SP)
        move.w  #2,-(SP)            * メモリの上方向から
        FNC     $7D                 * malloc2
        addq.l  #6,SP               * ダウンロードバッファを確保
        tst.l   D0
        bmi     alloc_err

        move.l  D0,memptr

        move.l  #1,-(SP)            * no sleep
        pea     buff                * message buffer
        pea     main                * entry
        move.w  SR,-(SP)            * SR
        pea     _SSP
        pea     _USP
        move.w  #8,-(SP)            * level
        pea     bpname
        FNC     $F8                 * newthread
        lea     $1C(SP),SP
        tst.l   D0                  * エラー？
        bmi     illegal             *   ならばillegalへ

        move.l  #3,-(SP)            * len
        move.l  memlen,-(SP)        * alllen
        move.l  memptr,-(SP)        * memptr
        move.w  D0,-(SP)            * thread #
        FNC     $7F                 * suballoc
        lea     14(SP),SP
        move.l  D0,A0               * ライン0の内容
        move.b  #13,(A0)+           * CR
        move.b  #10,(A0)+           * LF
        clr.b   (A0)                * eol

        pea     keep_msg
        FNC     $09                 * print
        addq.l  #4,SP

        clr.w   -(SP)
        move.l  #BOTTOM-start,-(SP)
        FNC     $31                 * keeppr
```

```
ver_err:
                pea     ver_err_msg
                bra     print_exit
module_err:
                pea     module_err_msg
                bra     print_exit
alloc_err:
                pea     alloc_err_msg
                bra     print_exit
illegal:
                pea     illegal_msg
print_exit:
                FNC     $09             * print
                addq.l  #4,SP
__exit:
                FNC     $00             * exit

                .data
name:
                dc.b    'ABDP.X',0

keep_msg:
                dc.b    'ABD COREが常駐しました',13,10
                dc.b    0
ver_err_msg:
                dc.b    'Humanのバージョンが違います。',13,10
                dc.b    0
module_err_msg:
                dc.b    'モジュール"ABDP.X"がみつかりません。',13,10
                dc.b    0
alloc_err_msg:
                dc.b    '必要なだけのメモリを確保できませんでした。',13,10
                dc.b    0
illegal_msg:
                dc.b    'スレッドを作成できませんでした。',13,10
                dc.b    0

                .bss
                .even
PSP_plus_10:
                ds.l    1
cmdline:
                ds.l    1
name0:
                ds.b    128

*       ★★★ 以下は新しいスレッドによって動作 ★★★

                text
main:
                clr.l   line_counter    * ポインタ、フラグの初期化
                move.w  #TRUE,down_flag
                move.l  #line_buffer,bufptr
                clr.l   line_length
                move.l  memptr,now_ptr

                bra     disp_stat

main_loop:
                FNC     $FF             * juggle

                tst.w   buff+10         * メッセージは届いているか?
                bpl     message         *   届いていればmassageへ
down:
                tst.w   down_flag       * ダウンロード中?
*               beq     main_loop       *   でなければmain_loopへ (処理が軽い)
                beq     disp_stat       *   でなければdisp_statへ (表示が消えない)
                                        *    ↑お好みの方をどうぞ

                IOCS    $33             * ISNS232C (IOCS)
                tst.l   D0              * 受信データあり?
*               beq     main_loop       *   ないならmain_loopへ (処理が軽い)
                beq     disp_stat       *   ないならdisp_statへ (表示が消えない)
                                        *    ↑お好みの方をどうぞ
                IOCS    $32             * INP232C (IOCS)
                move.l  bufptr,A0
                move.b  D0,(A0)+        * 1文字バッファへ
                move.l  A0,bufptr
                addq.l  #1,line_length
                cmp.b   #$0A,D0         * LF ?
```

```
		bne	main_loop

		move.l	line_length,D0
		addq.l	#1,D0
		move.l	D0,-(SP)
		FNC	$48			* malloc
		addq.l	#4,SP
		tst.l	D0			* エラー?
		bmi	down_stop		*   ならばdown_stopへ

		move.l	D0,A0
		move.l	D0,now_ptr
		lea	line_buffer,A2
		move.l	line_length,D0
		subq	#1,D0
copy_loop:
		move.b	(A2)+,(A0)+		* 非効率的..
		dbra	D0,copy_loop
		move.b	#0,(A0)+

		addq.l	#1,line_counter
		move.l	#line_buffer,bufptr
		clr.l	line_length
disp_stat:
		move.l	now_ptr,D0
		sub.l	memptr,D0
		add.l	line_length,D0
		moveq	#100,D1
		FEFUNC	$00			* __LMUL
		move.l	memlen,D1
		FEFUNC	$01			* __LDIV
		moveq	#3,D1			* 3桁
		lea	percent,A0
		FEFUNC	$18			* __IUSING
		move.b	#'%',eol1
		move.l	line_counter,D0
		moveq	#5,D1			* 5桁
		lea	line,A0
		FEFUNC	$18			* __IUSING
		moveq	#%1_1_01,D1		* 水色リバース強調
		moveq	#0,D2
		moveq	#0,D3			* (0,0)
		moveq	#17-1,D4		* 17文字
		lea	status_msg,A1
		IOCS	$2F			* B_PUTMES (IOCS)

		tst.w	down_flag
		beq	disp_stat1

		lea	down_msg,A1
		bra	disp_stat2
disp_stat1:
		lea	no_down_msg,A1
disp_stat2:
		moveq	#%1_1_01,D1		* 水色リバース強調
		moveq	#17,D2
		moveq	#0,D3			* (17,0)
		moveq	#12-1,D4		* 12文字
		IOCS	$2F			* B_PUTMES (IOCS)

		bra	main_loop

message:
		move.w	buff+8,D0		* アトリビュート取得
		beq	init			*   0ならinitへ
		subq.w	#2,D0
		bmi	list_start		*   1ならlist_startへ
		beq	list_cont		*   2ならlist_contへ
		subq.w	#2,D0
		bmi	flush			*   3ならflushへ
		beq	down_stop		*   4ならdown_stopへ
		subq.w	#2,D0
		bmi	down_cont		*   5ならdown_contへ
		beq	release			*   6ならreleaseへ

		bra	restore			* それ以外は無視

init:
*		仕様変更の結果、使用せず
		bra	restore
```

```
list_start:
		move.l	body,D1			* 開始ライン
		move.l	body+4,last_line	* 終了ライン

		move.l	line_counter,D0
		cmp.l	D1,D0			* 開始行>現在行?
		bcs	list_err		*   ならばlist_errへ

		move.l	memptr,A0
		move.l	D1,list_counter
		beq	list1
		subq	#1,D1
list_loop1:
		move.l	12(A0),A0		* next block
		dbra	D1,list_loop1
list1:
		bra	list_cont1

list_cont:
		move.l	next_line_ptr,A0
list_cont1:
		move.l	list_counter,D1
		move.l	12(A0),D0		* 次のラインのメモリブロック
		beq	list_cont2
		cmp.l	last_line,D1
		bcs	list_cont3
list_cont2:
		moveq	#-1,D1
list_cont3:
		move.l	D0,next_line_ptr
		lea	16(A0),A0
		move.l	D1,common_buffer
		move.l	A0,common_buffer+4
list_cont4:
		move.l	#8,-(SP)		* length
		pea	common_buffer
		clr.l	-(SP)			* ofs
		pea	common_name
		move.w	#2,-(SP)		* WRITE
		FNC	$55			* common
		lea	18(SP),SP

		addq.l	#1,list_counter

		bra	restore
list_err:
		move.l	#-2,common_buffer
		move.l	#list_err_msg,common_buffer+4

		bra	list_cont4
flush:
		clr.l	-(SP)			* 確保した全ての領域を開放
		FNC	$49			* mfree
		addq.l	#4,SP

		clr.l	line_counter
		move.w	#TRUE,down_flag
		move.l	#line_buffer,bufptr
		clr.l	line_length
		move.l	memptr,now_ptr

		bra	restore
down_stop:
		move.w	#FALSE,down_flag
		bra	restore
down_cont:
		move.w	#TRUE,down_flag
restore:
		move.w	#$FFFF,buff+10

		bra	disp_stat

release:
		clr.l	-(SP)			* 確保した全メモリを開放
		FNC	$49			* mfree
		addq.l	#4,SP
		move.l	body,A0			* 非常駐部に返事を返す
		move.l	memptr,(A0)		* 返事は、管理メモリ領域の管理ポインタ

		FNC	$F9			* kill
```

```
            .data
            .even
common_name:
            dc.b    'ABD_LIST',0

bpname:
            dc.b    'ABD CORE V1.0',0
            .even
buff:
            dc.l    8               * length (固定)
            dc.l    body            * 本体へのポインタ(固定)
            dc.w    0               * アトリビュート
            dc.w    $ffff           * 送信スレッド#

status_msg:
            dc.b    '('
percent:
            ds.b    3
eol1:
            dc.b    '%) LINE
line:
            ds.b    5+1
down_msg:
            dc.b    '  DOWNLOAD中',0
no_down_msg:
            dc.b    '  一時停止中',0

list_err_msg:
            dc.b    '範囲の指定に誤りがあります。',0

            .bss
            .even
down_flag:
            ds.w    1

body:
            ds.b    8
memlen:
            ds.l    1
memptr:
            ds.l    1
list_counter:
            ds.l    1
next_line_ptr:
            ds.l    1
last_line:
            ds.l    1
common_buffer:
            ds.l    2
line_counter:
            ds.l    1
now_ptr:
            ds.l    1
bufptr:
            ds.l    1
line_length:
            ds.l    1
line_buffer:
            ds.b    1024

            .stack
            .even
            ds.b    1024
_USP:
            ds.b    2048
_SSP:

BOTTOM:                                     * BSS/STACK領域はプログラムの一番後ろに
                                            * リロケートされるため、ラベル"BOTTOM"は
                                            * 常駐部の最後尾を指すことになる。

            .end

*   このモジュールは２つの部分から構成されています。
* ★ブートアップ／非常駐部
*   ここでは、主にユーザーインターフェースを行うモジュール"ABDP.X"を呼
* び出しを行い、そのリターンコードに従って、ただ非常駐終了するか、もし
* くはダウンロードバッファを確保して新しいスレッドを作成します。
* ★常駐部
*   このプログラムの核となる部分で、"ABD CORE"と呼んでいます。
*   この部分は新しく作成されたスレッドによって動作します。
```

```
*
*    このモジュールはメモリに常駐することを考え、なるべくメモリ占有量が
* 小さくなるよう、意識して作りました。
* （もっとも、スタック領域が大きいのであまり意味はありませんが）
*
* ［注目点］
* ★オーバレイXファイルのモジュール呼び出し
* ★バックグラウンド関連ファンクションの利用
* ★管理メモリ領域の利用
* ★メッセージ、コモンエリアによるプロセス間通信
```

ABDP.S

```
*
*        Automatic Background Downloader Ver1.00
*                                         (Human2.0x Only)
*        Created 1989 by M.Yoshizawa
*
*        モジュール#1  "ABDP.X" (ユーザーインターフェースモジュール)
*

                include macro.h

FEFUNC          macro   number
                dc.b    $FE
                dc.b    number
                endm

                .text
start:
                addq.l  #1,A2
                move.l  A2,cmdline

                pea     title
                FNC     $09             * print
                addq.l  #4,SP

                move.w  #TRUE,exist_flag * exist_flagを立てておく

                pea     my_bpdb_buf     * 自分のスレッド情報を得る
                move.w  #-2,-(SP)       * MINE
                FNC     $FA             * getthread
                addq.l  #6,SP

                move.w  D0,my_thread    * 自分のスレッド番号をmy_threadへ

                pea     bpdb_buf        * スレッド存在チェック
                move.w  #-1,-(SP)       * CHECK
                FNC     $FA             * getthread
                addq.l  #6,SP
                tst.l   D0              * 存在する？
                bpl     exist           *   ならばexistへ

                move.w  #FALSE,exist_flag * exist_flagを降ろす
exist:
                move.w  D0,his_thread   * 常駐部のスレッド番号をhis_threadへ
                move.w  #TRUE,nocmd_flag
cmd_loop:
                move.l  cmdline,A2

                lea     arg,A1
                bsr     getarg          * arg取得ルーチンコール
                tst.l   D0              * argの文字数は０？
                beq     cmd_eol         *   ならばcmd_eolへ

                move.l  A2,cmdline

                lea     arg,A1
                move.b  (A1)+,D0        * １文字取ってくる
*               beq     cmd_eol         *   行末コードならcmd_eolへ 《あり得ない》
                cmp.b   #'-',D0         * ハイフン？
                beq     option          *   ならばoptionへ
                cmp.b   #'/',D0         * スラッシュ？
                bne     send_string     *   でもなければsend_stringへ
option:
                move.b  (A1)+,D0        * もう１文字取ってくる
                beq     cmd_eol         *   行末コードならcmd_eolへ
```

```
        or.b    #$20,D0         * 大文字→小文字
        cmp.b   #'b',D0         * -B : バッファ容量の指定？
        beq     _bufsize        *   ならば_bufsizeへ
        cmp.b   #'l',D0         * -L : ダウンロードバッファ表示？
        beq     _list           *   ならば_listへ
        cmp.b   #'v',D0         * -V : ダウンロードバッファ表示２？
        beq     _view           *   ならば_viewへ
        cmp.b   #'f',D0         * -F : ダウンロードバッファフラッシュ？
        beq     _flush          *   ならば_flushへ
        cmp.b   #'s',D0         * -S : ダウンロード一時停止？
        beq     _stop           *   ならば_stopへ
        cmp.b   #'c',D0         * -C : ダウンロード再開？
        beq     _cont           *   ならば_contへ
        cmp.b   #'p',D0         * -P : スレッドのサスペンド／サスペンド解除？
        beq     _s_flip         *   ならば_s_flipへ
        cmp.b   #'r',D0         * -R : スレッド削除？
        beq     _release        *   ならば_releaseへ

        bra     cmd_err         * これら以外ならcmd_errへ

_bufsize:
        tst.w   exist_flag      * すでにスレッドは存在する？
        bne     exist_err       *   ならばexist_errへ

        move.l  A1,A0
        FEFUNC  $10             * _STOL
        bcs     bufsize_err     *   エラーならばbufsize_errへ
        cmp.l   #32,D0          * ３２以上？
        bcs     bufsize_err     *   でなければbufsize_errへ
        cmp.l   #256,D0         * ２５６未満？
        bcc     bufsize_err     *   でなければbufsize_errへ
        move.l  D0,alloc_size   * mallocすべきサイズをalloc_sizeへ

        bra     cmd_loop

_list:                          * ダウンロードバッファ表示
        move.w  #FALSE,view_flag
_list1:
        tst.w   exist_flag      * すでにスレッドは存在する？
        beq     not_exist_err   *   存在しないならばnot_exist_errへ

        bsr     getarea         * 領域解析ルーチンコール
        tst.l   D1              * 領域指定にミスがある？
        bmi     area_err        *   ならばarea_errへ

        pea     common_name     * 有ろうが無かろうがお構い無し
        move.w  #5,-(SP)        * DELETE
        FNC     $55             * common
        addq.l  #6,SP           * エラーも平気

        move.l  #-1,now_ptr     * 現在の読みだし行ポインタを初期化
        move.l  D1,now_ptr2

        move.l  D1,msg_body     * 開始行をmsg_bopyに
        move.l  D2,msg_body+4   * 終了行をmsg_body+4に
        moveq.l #1,D1           * 1 : LIST開始コマンド
        bsr     msg             * メッセージ送信サブルーチンコール
_list_loop:
        move.l  #8,-(SP)        * length
        pea     getbuf
        clr.l   -(SP)           * ofs
        pea     common_name
        move.w  #1,-(SP)        * READ
        FNC     $55             * common
        lea     18(SP),SP
        tst.l   D0              * エラー？
        bpl     _list_disp      * でなければ_list_dispへ
_list_juggle:
        FNC     $FF             * juggle
        bra     _list_loop
_list_disp:
        move.l  getbuf,D1       * 現在行ナンバーをget
        bmi     _list_disp1

        cmp.l   now_ptr,D1      * こちら側のポインタと比較
        beq     _list_juggle    *   同じなら_list_juggleへ
        move.l  D1,now_ptr
        bra     _list_disp2
_list_disp1:
```

```
        cmp.l   #-2,D1
        beq     _list_disp3
_list_disp2:
        tst.w   view_flag
        beq     _list_disp3

        move.l  D1,D7                   * D1を保存
        move.l  now_ptr2,D0             * 行番号文字列を作成
        moveq   #5,D1                   * 5桁
        lea     line,A0
        FEFUNC  $18                     * __IUSING
        pea     line                    * 行番号表示
        FNC     $09                     * print
        addq.l  #4,SP
        pea     colon                   * コロンの表示
        FNC     $09                     * print
        addq.l  #4,SP
        addq.l  #1,now_ptr2
        move.l  D7,D1                   * D1を元に戻す
_list_disp3:
        move.l  getbuf+4,-(SP)          * 1ライン表示
        FNC     $09                     * print
        addq.l  #4,SP

        tst.l   D1
        bmi     _list_end

        moveq.l #2,D1                   * 2 : LIST継続コマンド
        bsr     msg                     * メッセージ送信サブルーチンコール

        bra     _list_loop
_list_end:
        pea     list_msg
        FNC     $09                     * print
        addq.l  #4,SP

        move.w  #FALSE,nocmd_flag

        bra     cmd_loop

_view:
                                        * ダウンロードバッファ表示（行番号付）
        move.w  #TRUE,view_flag
        bra     _list1

_flush:
                                        * ダウンロードバッファフラッシュ
        lea     flush_msg,A1
        moveq.l #3,D1                   * 3 : バッファフラッシュコマンド
        bra     send_msg

_stop:
                                        * ダウンロード一時停止
        lea     stop_msg,A1
        moveq.l #4,D1                   * 4 : ダウンロード一時停止コマンド
        bra     send_msg

_cont:
                                        * ダウンロード再開
        lea     cont_msg,A1
        moveq.l #5,D1                   * 5 : ダウンロード再開コマンド
        bra     send_msg

send_msg:
        tst.w   exist_flag              * すでにスレッドは存在する？
        beq     not_exist_err           *   存在しないならばnot_exist_errへ

        bsr     msg                     * メッセージ送信サブルーチンコール

        move.l  A1,-(SP)                * 各コマンドのメッセージを表示
        FNC     $09                     * print
        addq.l  #4,SP

        move.w  #FALSE,nocmd_flag

        bra     cmd_loop

_release:
                                        * スレッドから削除
        tst.w   exist_flag              * すでにスレッドは存在する？
        beq     not_exist_err           *   存在しないならばnot_exist_errへ

        move.l  #memptr,msg_body        * スレッド＃0で走っている場合
        move.l  #-1,memptr              * メッセージが送り返してもらえないため
                                        * 返事を返すアドレスを指定する。
```

```
        moveq.l #6,D1           * 6 : releaseコマンド
        bsr     msg
_release1:
        tst.l   memptr          * 返事は返ってきたか？
        bmi     _release1       *   まだなら待つ

        move.l  memptr,-(SP)    * 常駐部のためのメモリを開放
        FNC     $49             * mfree
        addq.l  #4,SP

        pea     release_msg
        FNC     $09             * print
        addq.l  #4,SP

        moveq   #1,D0
        bra     exit

_s_flip:                        * スレッドのサスペンド／サスペンド解除
        tst.w   exist_flag      * すでにスレッドは存在する？
        beq     not_exist_err   *   存在しないならばnot_exist_errへ

        move.w  #FALSE,nocmd_flag

        move.b  bpdb_buf+4,D0   * スレッドの状態を取得
        beq     suspend         *   0ならばsuspendへ
        cmp.b   #$FE,D0         * サスペンド状態？
        bne     cmd_loop        *   でなければcmd_loopへ戻る
wakeup:
        clr.l   -(SP)           * dammy
        clr.l   -(SP)           * dammy
        move.w  #$FFFB,-(SP)    * WAKEUP
        move.w  his_thread,-(SP)
        move.w  my_thread,-(SP)
        FNC     $FD             * message
        lea     14(SP),SP
        pea     wakeup_msg
        FNC     $09             * print
        addq.l  #4,SP

        bra     cmd_loop
suspend:
        move.w  his_thread,-(SP)
        FNC     $FB             * suspend
        addq.l  #2,SP
        pea     suspend_msg
        FNC     $09             * print
        addq.l  #4,SP

        bra     cmd_loop

send_string:
        move.w  #3,-(SP)        * 標準ＡＵＸ入出力へ
        pea     -1(A1)          * argの内容
        FNC     $1E             * fputs
        addq.l  #4,SP
        move.w  #FALSE,nocmd_flag

        bra     cmd_loop

cmd_eol:
        tst.w   exist_flag      * すでにスレッドは存在する？
        bne     not_exist_exit  *   ならばnot_exist_exitへ

        move.l  alloc_size,D0
        or.w    #$FF00,D0
        bra     exit

not_exist_exit:
        tst.w   nocmd_flag
        beq     not_exist_exit1

        pea     no_cmd_msg
        FNC     $09             * print
        addq.l  #4,SP
not_exist_exit1:
        moveq   #0,D0
exit:
        move.w  D0,-(SP)
        FNC     $4C             * exit2
```

```
cmd_err:
                pea     cmd_err_msg
                bra     disp_abort
exist_err:
                pea     exist_err_msg
                bra     disp_abort
bufsize_err:
                pea     bufsize_err_msg
                bra     disp_abort
not_exist_err:
                pea     not_exist_err_msg
                bra     disp_abort
area_err:
                pea     area_err_msg
                bra     disp_abort
msg_err:
                pea     msg_err_msg
disp_abort:
                FNC     $09                 * print
                addq.l  #4,SP

                FNC     $00                 * exit (RETURN CODE = 0)

msg:                                        * メッセージ送信サブルーチン
                move.l  #8,-(SP)            * メッセージ長
                pea     msg_body            * mesptr
                move.w  D1,-(SP)            * atr (この場合コマンドコード)
                move.w  his_thread,-(SP)    * receiver
                move.w  my_thread,-(SP)     * sender
                FNC     $FD                 * message
                lea     14(SP),SP
                cmp.l   #-28,D0             * 相手が受け取れない?
                bne     msg0                *   そうでなければmsgへ

                FNC     $FF                 * juggle
                bra     msg
msg0:
                tst.l   D0                  * それ以外のエラー?
                bmi     msg_err             *   ならばmsg_errへ(やや乱暴なやり方)

                rts

getarg:                                     * arg取得サブルーチン
                moveq   #0,D0               * 文字数リセット
                moveq   #FALSE,D6           * quoteフラグリセット
                moveq   #FALSE,D7           * arg開始フラグリセット
getarg_loop:
                move.b  (A2)+,D1            * 1文字get
                beq     eol                 *   $00ならばeolへブランチ
                cmp.b   #$22,D1             * ダブルクォート?
                beq     quote               *   ならばquoteへ
                cmp.b   #' ',D1             * スペース?
                beq     delim               *   ならばdelimへ
                cmp.b   #9,D1               * タブ?
                beq     delim               *   ならばdelimへ
other:                                      * それ以外ならば. .
                moveq   #TRUE,D7            *   arg開始フラグをセット
                move.b  D1,(A1)+            * argバッファにコピー & ポインタ+1
                addq.l  #1,D0               * 文字数+1
                bra     getarg_loop
quote:
                tst.l   D6                  * quote開始済み?
                bne     eol                 *   ならばeolへ
                moveq   #TRUE,D6
                moveq   #TRUE,D7
                bra     getarg_loop
delim:
                tst.l   D7                  * arg開始済み?
                beq     getarg_loop         *   まだならばgetarg_loopへ
                tst.l   D6                  * quote開始済み?
                bne     other               *   ならばotherへ
eol:
                subq.l  #1,A2               * コマンドラインポインタ補正
                clr.b   (A1)+               * argにエンドマーク

                rts

getarea:                                    * 範囲指定取得サブルーチン
```

```
        clr.l   D1
        clr.l   D2

        move.b  (A1),D0             * 最初の１文字
        beq     getarea4
        cmp.b   #'-',D0             * ハイフン？
        beq     getarea2            *   ならばgetarea2へ
        move.l  A1,A0
getarea0:
        move.b  (A1),D0
        beq     getarea3
        cmp.b   #'0',D0             * '0'未満？
        bcs     getarea1            *   ならばgetarea1へ
        cmp.b   #':',D0             * '9'より大きい？
        bcc     getarea1            *   ならばgetarea1へ
        addq.l  #1,A1
        bra     getarea0
getarea1:
        cmp.b   #'-',D0             * デリミタは'-'？
        bne     getarea_err         *   でなければgetarea_errへ

        clr.b   (A1)+
        FEFUNC  $10                 * __STOL
        bcs     getarea_err         *   エラーならgetarea_errへ

        move.l  D0,D1

        tst.b   (A1)                * '-'の後ろはeol？
        beq     getarea5            *   ならばgetarea5へ

        move.l  A1,A0
        FEFUNC  $10                 * __STOL
        bcs     getarea_err         *   エラーならgetarg_errへ

        move.l  D0,D2
        rts
getarea2:
        lea     1(A1),A0
        FEFUNC  $10                 * __STOL
        bcs     getarea_err         *   エラーならgetarg_errへ

        move.l  D0,D2

        rts
getarea3:
        FEFUNC  $10                 * __STOL
        bcs     getarea_err         *   エラーならgetarg_errへ

        move.l  D0,D1
        move.l  D0,D2

        rts
getarea4:
        move.l  #0,D1
        move.l  #-1,D2

        rts
getarea5:
        move.l  #-1,D2

        rts
getarea_err:
        move.l  #-1,D1
        move.l  #-1,D2

        rts

        .data
bpdb_buf:
        ds.b    $60
        dc.b    'ABD CORE V1.0',0
        ds.b    4
alloc_size:
        dc.l    64

common_name:
        dc.b    'ABD_LIST',0

title:
        dc.b    'Automatic Background Downloader Ver1.00',13,10
```

```
                dc.b    'Created 1989 by M.Yoshizawa',13,10
                dc.b    13,10
                dc.b    0
cmd_err_msg:
                dc.b    ' usage: ABD [<送信文字列>] [-B<size>] [-L<area>] [-V<area>] [-S] [-C] [-F]
[-P] [-R]',13,10
                dc.b    '       -B<size> : ダウンロードバッファのサイズを指定します。',13,10
                dc.b    '                  単位はKB。32～255の範囲で指定します。',13,10
                dc.b    '                  デフォルトは64KB',13,10
                dc.b    '       -L<area> : 指定した範囲のダウンロードバッファの内容を表示します。',
13,10
                dc.b    '       -V<area> : 指定した範囲のダウンロードバッファの内容を表示します(行
番号付き)。',13,10
                dc.b    '       -S       : ダウンロードを一時停止します。',13,10
                dc.b    '       -C       : ダウンロードを再開します。',13,10
                dc.b    '       -F       : ダウンロードバッファをフラッシュします。',13,10
                dc.b    '       -P       : ABD_COREのスレッドをサスペンド/サスペンド解除します。',
13,10
                dc.b    '       -R       : プログラムをスレッドから削除し、メモリを開放します。',13,
10
                dc.b    13,10
                dc.b    0
exist_err_msg:
                dc.b    'すでにABD COREはスレッドに登録されています。',13,10
                dc.b    0
bufsize_err_msg:
                dc.b    'バッファのサイズの指定に誤りがあります。',13,10
                dc.b    0
not_exist_err_msg:
                dc.b    'ABD COREがスレッドに登録されていません。',13,10
                dc.b    0
area_err_msg:
                dc.b    '範囲の指定に誤りがあります。',13,10
                dc.b    0
msg_err_msg:
                dc.b    'メッセージが送信できません。',13,10
                dc.b    0
list_msg:
                dc.b    13,10
                dc.b    0
flush_msg:
                dc.b    'ダウンロードバッファをフラッシュしました。',13,10
                dc.b    0
stop_msg:
                dc.b    'ダウンロードを一時停止します。',13,10
                dc.b    0
cont_msg:
                dc.b    'ダウンロードを再開します。',13,10
                dc.b    0
release_msg:
                dc.b    'ABD COREをスレッドから削除しました。',13,10
                dc.b    0
wakeup_msg:
                dc.b    'ABD COREのスレッドのサスペンドを解除しました。',13,10
                dc.b    0
suspend_msg:
                dc.b    'ABD COREのスレッドをサスペンド状態にしました。',13,10
                dc.b    0
no_cmd_msg:
                dc.b    '送信文字列/オプションを指定してください。',13,10
                dc.b    0
colon:
                dc.b    ' : ',0

                .bss
exist_flag:
                ds.w    1
nocmd_flag:
                ds.w    1
view_flag:
                ds.w    1

cmdline:
                ds.l    1
memptr:
                ds.l    1

now_ptr:
                ds.l    1
```

```
now_ptr2:
		ds.l	1

my_thread:
		ds.w	1
his_thread:
		ds.w	1
msg_body:
		ds.b	8
getbuf:
		ds.b	1024
my_bpdb_buf:
		ds.b	$74

line:
		ds.b	6
arg:
		ds.b	64

		.end

*　このモジュールは、ユーザーインターフェースのためのモジュールです。
* ここで行ったコマンドラインの解析の結果は、モジュール#0"ABD.X"へはリ
* ターンコードで、常駐している"ABD CORE"へはメッセージシステムで通知さ
* れます。また、ダウンロードバッファの表示ではコモンエリアによる通信も
* 行っています。
*　このモジュール部分はメモリに常駐しないため、多少大きくても問題あり
* ません。
*
* [注目点]
* ★メッセージ、コモンエリアによるプロセス間通信
* ★スレッドの消去と、その管理メモリ領域の開放
```

割り込みベクタ表

ベクタ番号	アドレス	内　容	処理先*
000($00)	$000000	リセットベクタ・SSPの初期値	ROM(未使用)
001(&01)	$000004	リセットベクタ・PCの初期値	ROM(未使用)
002($02)	$000008	バスエラー	Human
003($03)	$00000C	アドレスエラー	Human
004($04)	$000010	不当命令	Human
005($05)	$000014	0による除算	Human
006($06)	$000018	CHK命令	Human
007($07)	$00001C	TRAPV命令	Human
008($08)	$000020	特権違反	Human
009($09)	$000024	トレース例外処理	Human
010($0A)	$000028	line 1010 emulator	Human
011($0B)	$00002C	line 1111 emulator (ファンクションコール/FEファンクションコール)	Human/RAM
012($0C)	$000030	reserved	未使用
:	:	:	:
014($0E)	$000038	reserved	未使用
015($0F)	$00003C	アンイニシャライズド割り込み	未使用
016($10)	$000040	reserved	未使用
:	:	:	:
023($17)	$00005C	reserved	未使用
024($18)	$000060	スプリアス割り込み	未使用
025($19)	$000064	オートベクタ方式・レベル1割り込み	未使用
026($1A)	$000068	オートベクタ方式・レベル2割り込み	未使用
027($1B)	$00006C	オートベクタ方式・レベル3割り込み	未使用
028($1C)	$000070	オートベクタ方式・レベル4割り込み	未使用
029($1D)	$000074	オートベクタ方式・レベル5割り込み	未使用
030($1E)	$000078	オートベクタ方式・レベル6割り込み	未使用
031($1F)	$00007C	オートベクタ方式・レベル7割り込み (NMI)	Human
032($20)	$000080	TRAP #0	未使用
033($21)	$000084	TRAP #1	未使用
034($22)	$000088	TRAP #2	未使用
035($23)	$00008C	TRAP #3	未使用
036($24)	$000090	TRAP #4	未使用
037($25)	$000094	TRAP #5	未使用
038($26)	$000098	TRAP #6	未使用
039($27)	$00009C	TRAP #7	未使用
040($28)	$0000A0	TRAP #8 (ROMデバッガ用)	ROM/未使用
041($29)	$0000A4	TRAP #9 (db.x用)	RAM/未使用
042($2A)	$0000A8	TRAP #10 (リセット/フロントSW/外部SW/ソフトoff処理)	ROM
043($2B)	$0000AC	TRAP #11 (STOPキー処理)	Human
044($2C)	$0000B0	TRAP #12 (COPYキー処理)	Human
045($2D)	$0000B4	TRAP #13 (CTRL-C処理)	Human
046($2E)	$0000B8	TRAP #14 (エラー表示)	Human
047($2F)	$0000BC	TRAP #15 (IOCS)	ROM
048($30)	$0000C0	reserved	未使用
:	:	:	:
063($3F)	$0000FC	reserved	未使用
(以下はユーザー割り込みベクタ)			
064($40)	$000100	MFP GPIP 0 (RTCアラーム/1Hz)	ROM
065($41)	$000104	MFP GPIP 1 (EXPWON信号)	ROM
066($42)	$000108	MFP GPIP 2 (フロント電源スイッチ)	ROM
067($43)	$00010C	MFP GPIP 3 (FM音源)	ROM/Human
068($44)	$000110	MFP Timer-D	ROM/Human
069($45)	$000114	MFP Timer-C (マウスRTS/カーソル/FDパワー, etc.)	ROM
070($46)	$000118	MFP GPIP 4 (V-DISP)	ROM

ベクタ番号	アドレス	内容	処理先*
071($47)	$00011C	MFP GPIP 5（RTC クロック）	ROM
072($48)	$000120	MFP Timer-B（USART シリアルクロック）	ROM
073($49)	$000124	MFP Transmit Error（キーデータの送信エラー）	ROM
074($4A)	$000128	MFP Transmit Buffer Empty（キーデータ送信可）	ROM
075($4B)	$00012C	MFP Receive Error（キーデータの受信エラー）	ROM
076($4C)	$000130	MFP Recieve Buffer Full（キーデータの受信データあり）	ROM
077($4D)	$000134	MFP Timer-A（CTRC の V-DISP 信号カウント）	ROM
078($4E)	$000138	MFP GPIP 6（CRTC の指定ラスター）	ROM
079($4F)	$00013C	MFP GPIP 7（CRTC の H- SYNC 信号）	ROM
080($50)	$000140	SCCB	未使用
081($51)	$000144	SCCB	未使用
082($52)	$000148	SCCB（マウス 外部ステータス変化）	未使用
083($53)	$00014C	SCCB（マウス 外部ステータス変化）	未使用
084($54)	$000150	SCCB（マウス 1バイト入力）	ROM
085($55)	$000154	SCCB（マウス 1バイト入力）	ROM
086($56)	$000158	SCCB（マウス 特別受信条件）	未使用
087($57)	$00015C	SCCB（マウス 特別受信条件）	未使用
088($58)	$000160	SCCA（RS232C 送信バッファ空）	未使用
089($59)	$000164	SCCA（RS232C 送信バッファ空）	未使用
090($5A)	$000168	SCCA（RS232C 外部ステータス変化）	未使用
091($5B)	$00016C	SCCA（RS232C 外部ステータス変化）	未使用
092($5C)	$000170	SCCA（RS232C 1バイト入力）	ROM
093($5D)	$000174	SCCA（RS232C 1バイト入力）	ROM
094($5E)	$000178	SCCA（RS232C 特別受信条件）	未使用
095($5F)	$00017C	SCCA（RS232C 特別受信条件）	未使用
096($60)	$000180	FDCINT（2HD ディスクのステータス割り込み）	ROM
097($61)	$000184	FDDINT（2HD ディスクの挿入／イジェクト割り込み）	ROM
098($62)	$000188	HDCINT（ハードディスクのステータス割り込み）	ROM
099($63)	$00018C	PRNINT（プリンタのレディー割り込み）	Human
100($64)	$000190	DMAC CH. A 転送終了（2HD 用）	ROM
101($65)	$000194	DMAC CH. A エラー	ROM
102($66)	$000198	DMAC CH. B 転送終了（HDD 用）	ROM
103($67)	$00019C	DMAC CH. B エラー	ROM
104($68)	$0001A0	DMAC CH. C 転送終了（メモリ to メモリ用）	ROM
105($69)	$0001A4	DMAC CH. C エラー	ROM
106($6A)	$0001A8	DMAC CH. D 転送終了（ADPCM 用）	ROM
107($6B)	$0001AC	DMAC CH. D エラー	ROM
108($6C)	$0001B0	空き	未使用
:	:		
255($FF)	$0003FC	空き	未使用

処理先としては，標準と思われるシステム構成で Human を立ち上げた直後の状態のベクタの内容を示しています.

Human：処理先は Human のシステム領域（Human. sys＋デバイスドライバ等）

ROM　：処理先は ROM

RAM　：処理先は Human 以外の RAM

未使用　：Human 内のエラー処理ルーチンまたは ROM 内の rte を指している

SRAMの内容

X68000は$ED0000から$ED3FFFのアドレスがバッテリー・バックアップされたSRAMになっています。このアドレスには次のようなデータが書き込まれています。

アドレス	内容
$ED0000-$ED0007	メモリ・チェック・データとして次のようなデータが書き込まれている（このデータが壊れている場合、SRAMのデータはすべて初期化される）。 dc. b 'X68000', $57 なお「X」は全角文字、他は半角文字
$ED0008. l	メイン・メモリ最終アドレス＋1
$ED000C. l	ROMから起動する場合のジャンプ・アドレス
$ED010. l	SRAMから起動する場合のジャンプ・アドレス
$ED0014. l	アラームで起動した場合の起動している時間（分単位） －1の場合は無限大になる
$ED0018. w	最優先ブート・デバイス $0000 STD $8n00 HDn $9n00 2HDn $A000 ROM $B000 RAM
$ED001A. w	RS-232Cの設定 bit $E-$F ストップ・ビット $F $E 0 0 2 0 1 1 1 0 1.5 1 1 2 bit $C-$D パリティ $D $C 0 0 パリティなし 0 1 奇数パリティ 1 0 パリティなし 1 1 偶数パリティ bit $A-$B ビット長 $B $A 0 0 5ビット 0 1 6ビット 1 0 7ビット 1 1 8ビット bit 0-2 ボーレート 000 75 001 150 010 300 011 600 100 1200 101 2400 110 4800 111 9600
$ED001C. b	キーボードLEDの状態
$ED001C. b	bit 7 0にする bit 6 全角 bit 5 ひらがな bit 4 INS bit 3 CAPS bit 2 コード入力 bit 1 ローマ字 bit 0 かな
$ED001D. b	IPLにおける画面モードナンバー（IOCS$10のD1の値）
$ED001E. l	アラームで起動した場合に行なう行動（IOCS$5EのA1の値）
$ED0022. l	アラームで起動する時刻（IOCS$5EのD1の値）
$ED0026. b	アラーム禁止・許可フラグ（0:許可, 7:禁止）
$ED0027. b	OPT. 2キーによるTVコントロールフラグ（0:TVコントロール可能, 1:TVコントロール不可）
$ED0028. b	コントラスト・システム値
$ED0019. b 29	本体OFF時のディスク・イジェクト・フラグ（0:イジェクトしない, 1:イジェクトする）
$ED002A. b	本体OFF時のTVコントロール内容（IOCS$0CのD1の値）
$ED002B. b	キーボード仮名配列（0:JIS, 1:あいうえお順）
$ED002C. b	電卓の文字フォント （0:LCDフォント, 1:ノーマルフォント）
$ED002D. b	SRAMの使用モード 0:使っていない 1:SRAMDISKとして使用 2:プログラム・エリア
$ED002E. w	テキストパレット0のシステム値
$ED0030. w	テキストパレット1のシステム値
$ED0032. w	テキストパレット2のシステム値
$ED0034. w	テキストパレット3のシステム値
$ED0036. w	テキストパレット4~7のシステム値
$ED0038. w	テキストパレット8~15のシステム値
$ED003A. b	キー・リピート開始までの間隔（IOCS$08のD1の値）
$ED003B. b	キー・リピート間隔（IOCS$09のD1の値）
$ED003C. l	プリンタ・タイムアウト時間
$ED0040. l	SRAMが初期化されてからの稼働時間合計（分単位）
$ED0044. l	SRAMが初期化されてからの起動回数
$ED0048-$ED0058	システム予約
$ED0059. b	文字変更フラグ bit 0 0のとき¥, 1のとき／ bit 1 0のとき￣, 1のとき～ bit 2 0のとき‖, 1のとき¦
$ED005A. b	ハードディスク接続台数
$ED005B-$ED00FF	リザーブ
$ED0100-$ED03FF	プログラム・エリア
$ED0400-$ED3FFF	SRAMDISKエリア

SRAM起動プログラム

X68000はSRAMを装備しているため，リセット（または電源投入）したときにSRAMに書き込んだプログラムを起動することが可能になっています．また，このようなプログラムは次のようにすると簡単に作ることができます．

1：アセンブラで次の形式のプログラムを作る．

```
        bra     mainprog        プログラム先頭にくるようにする．

mainprog
        プログラム本体
        ここへジャンプしてくるときはスーパーバイザ・モードになっている．
        サービス・ルーチンはIOCSコールしか使えない．
        最後にRTS をつける（この場合レジスタは元に戻す）とSWITCH＝STDと見なされて優先順
        位にしたがって他のデバイスから起動する（再びSRAMから起動しようとする場合がある）．
```

2：アセンブルする．
3：－Bオプションを付けてリンクする．

LK －B ED0C40 ＊.0

実行するときは次のようにします．

4：SRAMDISKを初期化して，プログラム（＊.X）をコピーする．
5：SWITCH B＝RAM1とする（優先起動デバイスが変更される）．

この後RESETするとSRAMのプログラムから起動します．暴走した場合は OPT.1 を押しながらRESETして，2HDドライブ0から起動させてください．

取り除くときは，次のようにします．

6：SRAMDISKを初期化する．
7：文書ファイルをSRAMDISKにコピーする．
8：SWITCH B＝STDとする（またはHD0など）．

文書ファイルをコピーするのはプログラムを完全に消すためです．初期化・FORMATだけではプログラムが残るため，起動デバイスが用意されていない場合はSRAMのプログラムが実行されてしまう場合があります．「ディスクをセットしてください」表示が出たときは，すでにすべてのデバイスをチェックしてあるのでSRAMのプログラムが実行されている可能性があります．

サンプル・プログラムは以下のとおりです．

```
*
*
*       ＳＲＡＭ起動プログラム
*
*
*               起動すると、メッセージを表示します。
*

                bra     mainprog
                nop

mainprog
                movem.l D0/A1,-(SP)
                lea     mes(PC),A1
                moveq   #$21,D0
                trap    #15
                movem.l (SP)+,D0/A1
                rts

mes
                dc.b    'ＳＲＡＭ起動しました。',0
```

SRAM 起動解除プログラム

SRAM起動を完全に解除するのは面倒です．接ぎ木プログラムを実行すると，RAM&ROMからの起動は完全に解除されます．
プログラムは以下のとおりです．

```
*
*
*       SRAM Boot Cut Program
*
*

                moveq   #$86,D0
                move.b  #$31,D1
                lea     $E8E00D,A1
                trap    #15

                moveq   #$88,D0
                move.l  #$BFFFFC,D1
                lea     $ED000C,A1
                trap    #15

                moveq   #$88,D0
                move.l  #$ED0000,D1
                lea     $ED0010,A1
                trap    #15

                moveq   #$86,D0
                clr.b   D1
                lea     $E8E00D,A1
                trap    #15

                dc.w    $FF00
```

常駐型プログラムの作成方法

Human68K上には，実行するとプログラムの一部が常駐し，再度実行すると常駐部分が開放されるといったプログラムがあります（HISTORY.Xなど）．このようなプログラムは次のようなアリゴリズムで作成できます．

1：実行されたらメモリ管理ポインタをさかのぼってそれまでに常駐した「自分があるかを調べる．
2：「自分」がなければファンクション・コールの$FF31を使って常駐する．
3：「自分」があれば以前のプログラムが使っているメモリを開放し，常駐せずに終了する．

ただし，一般に常駐型プログラムは割り込みなどのベクタを書き換えることが多いので，そのベクタをもとに戻す処理も必要です．また，元に戻すために必要なデータの受け渡しも考慮しておかなくてはならないでしょう．

サンプル・プログラムは以下のとおりです．

```
*
*
*       常駐型プログラム例
*
*

*       このプログラムは「1つ」前のプログラムしかチェックしない

startadd
check
                dc.b    '判別データー'

prog
                movea.l (A0),A1                 * 1つ前のメモリブロックアドレス
                adda.l  #$100,A1                * 1つ前のプログラムの先頭アドレス
                lea     check(PC),A2            * 「自分」かどうかを調べる判断材料のアドレス
                moveq   #12-1,D0                * データーの長さ（DBRAでループさせるので
                                                * －1する）
loop
                cmpm.b  (A1)+,(A2)+             * 比較
                bne     start                   * 違いがあれば「自分」ではない
                dbra    D0,loop

                                                * すべて一致すれば「自分」である
                move.l  (A0),D0                 * 常駐していたメモリを開放
                addi.l  #$10,D0
                move.l  D0,-(SP)
                dc.w    $FF49
                addq.l  #4,SP

                dc.w    $FF00

start
                clr.w   -(SP)                   * 常駐する
                move.l  #endadd-startadd+1,-(SP)
                dc.w    $FF31

endadd
                .end    prog
```

ファンクションコール・エラーコード表

エラーコード	意　　味
−1　($FFFFFFFF)	無効なファンクションコールを実行しました
−2　($FFFFFFFE)	指定したファクション名はみつかりません
−3　($FFFFFFFD)	指定したディレクトリはみつかりません
−4　($FFFFFFFC)	オープンしているファイルが多すぎます
−5　($FFFFFFFB)	ディレクトリやボリュームラベルはアクセスできません
−6　($FFFFFFFA)	指定されたハンドルはオープンされていません
−7　($FFFFFFF9)	メモリ管理領域が壊されました
−8　($FFFFFFF8)	実行するのに必要なメモリがありません
−9　($FFFFFFF7)	無効なメモリ管理ポインタを指定しました
−10　($FFFFFFF6)	不正な環境を指定しました
−11　($FFFFFFF5)	実行ファイルのフォーマットが異常です
−12　($FFFFFFF4)	オープンでアクセスモードが異常です
−13　($FFFFFFF3)	ファイル名の指定で誤りがあります
−14　($FFFFFFF2)	無効なパラメーマでコールしました
−15　($FFFFFFF1)	ドライブ指定に誤りがあります
−16　($FFFFFFF0)	カレントディレクトリは削除できません
−17　($FFFFFFEF)	IOCTRL できないデバイスです
−18　($FFFFFFEE)	これ以上ファイルはみつかりません
−19　($FFFFFFED)	このファイルは書き込みができません
−20　($FFFFFFEC)	このディレクトリはすでに存在します
−21　($FFFFFFEB)	(ディレクトリ中に)ファイルがあるので削除できません
−22　($FFFFFFEA)	(デイレクトリ中に)ファイルがあるのでリネームできません
−23　($FFFFFFE9)	ディスクがいっぱいでファイルが作れません
−24　($FFFFFFE8)	ディレクトリがいっぱいでファイルが作れません
−25　($FFFFFFE7)	指定の位置にはシークできません
−26　($FFFFFFE6)	スーパーバイザ状態で再びスーパーバイザ・モードを指定しました
(以下は Human2.0x 以降)	
−28　($FFFFFFE4)	すでに同名のスレッドが存在します
−29　($FFFFFFE3)	メッセージが受け取られませんでした
−30　($FFFFFFE2)	スレッド番号に異常があります
−32　($FFFFFFE0)	シェアリングを行なうファイル数をオーバーしました
−33　($FFFFFFDF)	ロック違反です

文法コード表

ｺｰﾄﾞ	品詞の種類	品詞の例／内容
0	―――	
1	動詞カ行五段活用	解－く
2	動詞ガ行五段活用	継－ぐ
3	動詞サ行五段活用	話－す
4	動詞タ行五段活用	持－つ
5	動詞ナ行五段活用	死－ぬ
6	動詞バ行五段活用	呼－ぶ
7	動詞マ行五段活用	進－む
8	動詞ラ行五段活用	切－る
9	動詞ワ行五段活用	言－う
10	動詞サ行変格活用	する
11	動詞カ行変格活用	来る
12	動詞一段活用	着－る　与え－る　（上一段活用、下一段活用）
13	形容詞	暖か－い　良－い（ク活用とシク活用は区別しない）
14	形容動詞	静か－だ
15	形容動詞複合名詞	「有能（な人）」のように形容動詞としても使う名詞
16	サ変複合名詞	「愛（する）」、「惨敗（する）」などのようにサ変複合動詞になる名詞
17	名詞	形容動詞の語幹やサ変複合動詞にならない名詞
18	単漢字	漢字の読みの登録
19	人名（姓）	人名の姓
20	人名（名）	人名の名
21	地名	地名
22	団体名	会社、法人、政党の名前など
23	物の名称	商品名など
24	数詞	「個」、「頭」など
25	数字	数字
26	接尾語	語の下につける語（「さ」、「ぽい」、「がる」など）
27	感動詞	感動の呼びかけ、語の受け答え（「おや」、「まあ」、「あら」など）
28	接続語	前の言葉と後ろの言葉をつなぐ語（「または」、「そこで」，「ところで」など）
29	副詞	活用語を修飾する語（「はっきり」、「たいへん」、「とても」など）
30	連体詞	非活用語を修飾する語（「たった」、「すべての」、「あの」など）

さくいん

✣あとがき

貧弱だと言われ続けてきたX68000の開発環境が最近になって急速に向上してきたのは，デベロッパ側もさることながら，ユーザー側の努力によるところが大きいように思えます．

なぜ，私たちはこうまでX68000に惹かれるのでしょうか．

私に関する限り，その理由のひとつに，発表当時のパーソナルコンピュータを取り巻く状況があったような気がします．当時，PC-9801シリーズの優勢がほぼ確定し，かといって目ぼしい対抗馬もなく，なしくずし的にPC-9801シリーズの寡占状況が進行していました．ご存知のとおり，PC-9801という機械は個性に乏しく，『パソコン』というよりはむしろ『ただの道具』と自らを規定しているようなイメージのある機械です．それがわかっているのに，他の選択肢がないに等しいため，PC-9801を選ばざるを得ないという，『パソコン』好きの私にとっては，やるせない状況だったわけです．

そんなころに，まったく突然に姿を現したのがX68000でした．いまさらその個性的なスペックを並べることもないでしょう．私が狂喜したのは言うまでもありません．まぎれもない，それは『パソコン』でした．いまだに私は，そのときの感動を忘れることができません．そして，いまだにこの機械に夢中というわけです．

X68000ユーザーの皆さんがすべてこうだとは思いませんが，多かれ少なかれ同じような思いを抱いてX68000に向っているのではないかと勝手に推測しています．

一般のユーザーでも，作ったプログラムを通信を通じて簡単に全国に広めることができる時代です．良質なプログラム1本1本が，その機種のソフト文化を支える直接の力になります．

私が，そして皆さんが愛してやまないX68000のソフト文化振興のために，この本が少しでもお役にたてれば幸いです．

（吉沢）

X68000は遊べるマシンです．

この本は，X68000で「アセンブラプログラムを作って遊ぶ」人のために書きました．

最近，パソコン界全体としては「アセンブラでプログラムを作って遊ぶ」人が少なくなってしまったようです．ところが，X68000のユーザーはそういう人がかなりの割合を占めているようです．

この本には，そのような人達が欲しいと思われる情報を載せました．

皆さんの「遊び」が，この本によって楽しく深くなることをお祈りします．

（市原）

プログラマーのためのX68000環境ハンドブック

1989年12月15日　初版発行
1990年 2 月 5 日　第 2 版第 1 刷発行

著　者　吉沢正敏
市原昌文
発行者　星　正明
発行所　株式会社 工 学 社
〒151 東京都渋谷区代々木1-37-1 ぜんらくビル
☎(03)375-5784(代)〔営業〕
☎(03)320-1218(代)〔編集〕振替 東京5-22510

定価3000円（本体2913円）

印刷：日出島

ISBN4-87593-157-3 C3055 P3000E

プログラマーのための

X68000環境ハンドブック

工学社

定価3000円(本体2913円)

ISBN4-87593-157-3 C3055 P3000